Minimizing message size in stochastic communication patterns: fast self-stabilizing protocols with 3 bits

Lucas Boczkowski, Amos Korman, Emanuele Natale

Minimizing Message Size in Stochastic Communication Patterns: Fast Self-Stabilizing Protocols with 3 bits*

Lucas Boczkowski † Amos Korman † Emanuele Natale ‡

Abstract

This paper considers the basic $\mathcal{PULL}$ model of communication, in which in each round, each agent extracts information from few randomly chosen agents. We seek to identify the smallest amount of information revealed in each interaction (message size) that nevertheless allows for efficient and robust computations of fundamental information dissemination tasks. We focus on the *Majority Bit Dissemination* problem that considers a population of n agents, with a designated subset of *source agents*. Each source agent holds an *input bit* and each agent holds an *output bit*. The goal is to let all agents converge their output bits on the most frequent input bit of the sources (the *majority bit*). Note that the particular case of a single source agent corresponds to the classical problem of *Broadcast* (also termed *Rumor Spreading*). We concentrate on the severe fault-tolerant context of *self-stabilization*, in which a correct configuration must be reached eventually, despite all agents starting the execution with arbitrary initial states. In particular, the specification of who is a source and what is its initial input bit may be set by an adversary.

We first design a general compiler which can essentially transform any self-stabilizing algorithm with a certain property (called "the *bitwise-independence property*") that uses ℓ-bits messages to one that uses only $\log \ell$-bits messages, while paying only a small penalty in the running time. By applying this compiler recursively we then obtain a self-stabilizing *Clock Synchronization* protocol, in which agents synchronize their clocks modulo some given integer T, within $\tilde{\mathcal{O}}(\log n \log T)$ rounds w.h.p., and using messages that contain 3 bits only.

We then employ the new Clock Synchronization tool to obtain a self-stabilizing Majority Bit Dissemination protocol which converges in $\tilde{\mathcal{O}}(\log n)$ time, w.h.p., on every initial configuration, provided that the ratio of sources supporting the minority opinion is bounded away from half. Moreover, this protocol also uses only 3 bits per interaction.

1 Introduction

1.1 Background and motivation Distributed systems composed of limited agents that interact in a stochastic fashion to jointly perform tasks are common in the natural world as well as in engineered systems. Examples include a wide range of insect populations [36], chemical reaction networks [17], and mobile sensor networks [3]. Such systems have been studied in various disciplines, including biology, physics, computer science and chemistry, while employing different mathematical and experimental tools.

*A preliminary version of this work appears as a 3-pages Brief Announcement in PODC 2016 [14] and as an extended abstract at SODA 2017 [15].

†IRIF, CNRS and University Paris Diderot, Paris, 75013, France. E-mail: {Lucas.Boczkowski,Amos.Korman}@irif.fr.

‡Max Planck Institute for Informatics, Saarbrücken, 66123 , Germany. E-mail: emanuele.natale@mpi-inf.mpg.de. This work has been partly done while the author was visiting the Simons Institute for the Theory of Computing.

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 648032).

From an algorithmic perspective, such complex systems share a number of computational challenges. Indeed, they all perform collectively in dynamically changing environments despite being composed of limited individuals that communicate through seemingly unpredictable, unreliable, and restricted interactions. Recently, there has been significant interest in understanding the computational limitations that are inherent to such systems, by abstracting some of their characteristics as distributed computing models, and analyzing them algorithmically [2, 3, 6, 11, 29, 33]. These models usually consider agents which are restricted in their memory and communication capacities, that interact independently and uniformly at random (u.a.r.). By now, our understanding of the computational power of such models is rather advanced. However, it is important to note that much of this progress has been made assuming non-faulty scenarios - a rather strong assumption when it comes to natural or sensor-based systems. For example, to synchronize actions between processors, many known distributed protocols rely on the assumption that processors know when the protocol is initiated. However, in systems composed of limited individuals that do not share a common time notion, and must react to a dynamically changing environment, it is often unclear how to achieve such conditions. To have a better understanding of such systems, it is desirable to identify the weakest computational models that still allow for both efficient as well as robust computations.

This paper concentrates on the basic $\mathcal{PULL}$ model of communication [21, 23, 24, 38], in which in each round, each agent can extract (pull) information from few other agents, chosen u.a.r. In the computer science discipline, this model, as well as its companion $\mathcal{PUSH}$ model, gained their popularity due to their simplicity and inherent robustness to different kinds of faults. Here, focusing more on the context of natural systems, we view the $\mathcal{PULL}$ model as an abstraction for communication in well-mixed scenarios, where agents can occasionally "observe" arbitrary other agents. We aim to identify the minimal model requirements with respect to achieving basic information dissemination tasks under conditions of increased uncertainty. As many natural systems appear to be more restricted by their communication abilities than by their memory capacities [35, 1, 32], our main focus is on understanding what can be computed while revealing as few bits per interaction as possible[1]. In dealing with such an existential question, we do not claim that our solution represents actual plausible strategies employed in nature, yet we believe that such mathematical results can be helpful in understanding the limitations of natural systems.

Self-stabilizing Bit Dissemination. Disseminating information from one or several sources to the rest of the population is one of the most fundamental building blocks in distributed computing [16, 18, 21, 23, 38], and an important primitive in natural systems [49, 47, 48]. Here, we focus on the *Majority Bit Dissemination* problem defined as follows. We consider a population of n agents. The population may contain multiple *source agents* which are specified by a designated bit in the state of an agent indicating whether the agent is a source or not. Each source agent holds a binary *input bit*, however, sources may not necessarily agree on their input bits. In addition, each agent holds a binary *output bit* (also called *opinion*). The goal of all agents is to converge their opinion on the majority bit among the initial input bits of the sources, termed b_{maj}. This problem aims to capture scenarios in which some individuals view themselves as informed, but some of these agents could also be wrong, or not up-to-date. Such situations are common in nature [20, 47] as well as in man-made systems. The number of sources is termed k. We do not assume that agents know the value k, or that sources know whether they are in the majority or minority (in terms of their input bit). For simplicity, to avoid

[1] We note that stochastic communication patterns such as $\mathcal{PULL}$ or $\mathcal{PUSH}$ are inherently sensitive to congestion issues. Indeed, in such models it is unclear how to simulate a protocol that uses large messages while using only small size messages. For example, the straightforward strategy of breaking a large message into small pieces and sequentially sending them one after another does not work, since one typically cannot make sure that the small messages reach the same destination. Hence, reducing the message size may have a profound impact on the running time, and perhaps even on the solvability of the problem at hand.

dealing with the case that the fraction of the majority input bit among sources is arbitrarily close to that of the minority input bit, we shall guarantee convergence only when the fraction of source agents holding the majority input bit is bounded away from 1/2.

The particular case where we are promised to have $k = 1$ is called Bit Dissemination, for short. In this case we have a single source agent that aims to disseminate its input bit b to the rest of the population, and there are no other sources introducing a conflicting opinion. Note that this problem has been studied extensively in different models under different names (e.g., *Broadcast* or *Rumor Spreading*). A classical example of Bit Dissemination considers the synchronous $\mathcal{PUSH}/\mathcal{PULL}$ communication model, where b can be propagated from the source to all other agents in $\mathcal{O}(\log n)$ rounds, by simply letting each uninformed agent copy it whenever it sees an informed agent [38]. The correctness of this protocol heavily relies on the absence of incorrect information held by agents. Such reliability however may be difficult to achieve in dynamic or unreliable conditions. For example, if the source is sensitive to an unstable environment, it may change its mind several times before stabilizing to its final opinion. Meanwhile, it may have already invoked several consecutive executions of the protocol with contradicting initial opinions, which may in turn "infect" other agents with the wrong opinion $1 - b$. If agents do not share a common time notion, it is unclear how to let infected agents distinguish their current wrong opinion from the more "fresh", correct opinion. To address such difficulty, we consider the context of *self-stabilization* [22], where agents must converge to a correct configuration from any initial configuration of states.

1.2 Technical difficulties and intuition Consider the Bit Dissemination problem (where we are guaranteed to have a single source agent). This particular case is already difficult in the self-stabilizing context if we are restricted to use $O(1)$ bits per interaction. As hinted above, a main difficulty lies in the fact that agents do not necessarily share a common time notion. Indeed, it is easy to see that if all agents share the same clock, then convergence can be achieved in $\mathcal{O}(\log n)$ time with high probability (w.h.p.), i.e, with a probability of at least $1 - n^{-\Omega(1)}$, and using two bits per interaction.

Self-stabilizing Bit Dissemination ($k = 1$) with 2 bits per interaction, assuming synchronized clocks. The source sets her output bit to be her input bit b. In addition to communicate its output bit b_u, each agent u stores and communicates a *certainty* bit c_u. Informally, having a certainty bit equal to 1 indicates that the agent is certain of the correctness of its output bit. The source's certainty bit is always set to 1. Whenever a non-source agent v observes u and sees the tuple (b_u, c_u), where $c_u = 1$, it copies the output and certainty bits of u (*i.e.*, sets $b_v = b_u$ and $c_v = 1$). In addition, all non-source agents count rounds, and reset their certainty bit to 0 simultaneously every $T = \mathcal{O}(\log n)$ rounds. The reset allows to get rid of "old" output bits that may result from applying the protocol before the source's output bit has stabilized. This way, from the first time a reset is applied after the source's output bit has stabilized, the correct source's output bit will propagate to all agents within T rounds, w.h.p. Note however, that if agents do not share a consistent notion of time they cannot reset their certainty bit to zero simultaneously. In such cases, it is unclear how to prevent agents that have just reset their certainty bit to 0 from being "infected" by "misleading" agents, namely, those that have the wrong output bit and certainty bit equal to 1.

Self-stabilizing Bit Dissemination ($k = 1$) with a single bit per interaction, assuming synchronized clocks. Under the assumption that all agents share the same clock, the following trick shows how to obtain convergence in $\mathcal{O}(\log n)$ time and using only a single bit per message, namely, the output bit. As before, the source sets her output bit to be her input bit b. Essentially, agents divide time into phases of some prescribed length $T = \mathcal{O}(\log n)$, each of them being further subdivided into 2 subphases of length $T/2$. In the first subphase of each phase, non-source agents are *sensitive* to opinion 0. This

means that whenever they see a 0 in the output bit of another agent, they turn their output bit to 0, but if they see 1 they ignore it. Then, in the second subphase of each phase, they do the opposite, namely they switch their output bit to 1 as soon as they see a 1 (see Figure 1). Consider the first phase starting after initialization. If $b = 0$ then within one complete subphase $[1, T/2]$, every output bit is 0 w.h.p., and remains there forever. Otherwise, if $b = 1$, when all agents go over a subphase $[T/2 + 1, T]$ all output bits are set to 1 w.h.p., and remain 1 forever. Note that a common time notion is required to achieve correctness.

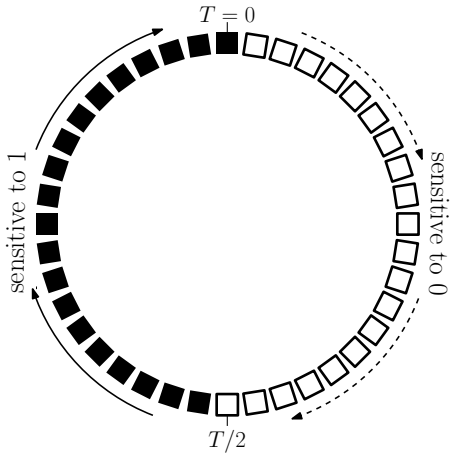

Figure 1: The division in subphases used for self-stabilizing Bit Dissemination with a clock. During the first half, between times 1 and $T/2$, agents are sensitive to 0. Then they are sensitive to 1.

The previous protocol indicates that the self-stabilizing Bit Dissemination problem is highly related to the self-stabilizing *Clock Synchronization* problem, where each agent internally stores a clock modulo $T = \mathcal{O}(\log n)$ incremented at every round and, despite having arbitrary initial states, all agents should converge on sharing the same value of the clock. Indeed, given such a protocol, one can obtain a self-stabilizing Bit Dissemination protocol by running the Clock Synchronization protocol in parallel to the last example protocol. This parallel execution costs only an additional bit to the message size and a $\mathcal{O}(\log n)$ additive term to the time complexity over the complexities of the Clock Synchronization protocol.

Intuition behind the self-stabilizing Clock Synchronization algorithm. Our technique for obtaining the Clock Synchronization protocol is based on a compact recursive use of the stabilizing consensus protocol proposed by Doerr et al. [24] through our Message Reduction Theorem (Theorem 3.1).

In the Preliminary section (Section 2.2) we describe a simple protocol called Syn-Simple that uses $\mathcal{O}(\log T)$ bits per message. In Syn-Simple, each agent u maintains a clock $C_u \in [0, T-1]$. At each round, each agent u displays the opinion of her clock, pulls 2 other such clock opinions, and updates her clock as the bitwise majority of the two clocks she pulled and her own. Then the clock C_u is incremented. This protocol essentially amounts to running the protocol of Doerr et al. on each bit separately and in parallel, and self-stabilizes in $\mathcal{O}(\log T \log n)$ rounds w.h.p. (Proposition 2.1).

We want to apply a strategy similar to Syn-Simple, while using only $\mathcal{O}(1)$ many bits per interaction. The core technical ingredient, made rigorous in the Message Reduction Theorem, is that a certain class of protocols using messages of ℓ bits, to which Syn-Simple belongs, can be emulated by another protocol which uses $\lceil \log \ell \rceil + 1$ bits only. The idea is to build a clock modulo ℓ using Syn-Simple itself on $\lceil \log \ell \rceil$ bits and sequentially display one bit of the original ℓ-bit message according to such clock. Thus,

by applying such strategy to Syn-Simple itself, we use a smaller clock modulo $\ell' \ll \ell$ to synchronize a clock modulo ℓ. Iterating such process, in Section 4.2, we obtain a compact protocol which uses only 3 bits.

1.3 The model

The communication model. We adopt the synchronous $\mathcal{PULL}$ model [12, 21]. Specifically, in the $\mathcal{PULL}(\eta)$ model, communication proceeds in discrete rounds. In each round, each agent u "observes" η arbitrary other agents, chosen uniformly at random (with replacement), which we abbreviate as u.a.r., among all agents, including herself. (We often omit the parameter η when it is equal to 2). When an agent u "observes" another agent v, she can peek into a designated *visible part* of v's memory. If several agents observe an agent v at the same round then they all see the same visible part. The *message size* denotes the number of bits stored in the visible part of an agent. We denote with $\mathcal{PULL}(\eta, \ell)$ the $\mathcal{PULL}(\eta)$ model with message size ℓ. We are primarily interested in message size that is independent of n, the number of agents.

Agents. We assume that agents do not have unique identities, that is, the system is *anonymous*. We do not aim to minimize the (non-visible) memory requirement of the agent, yet, we note that our constructions can be implemented with relatively short memory, using $\mathcal{O}(\log\log n)$ bits. We assume that each agent internally stores a clock modulo some integer $T = \mathcal{O}(\log n)$, which is incremented at every round.

Majority Bit Dissemination problem. We assume a system of n agents each having an internal state that contains an *indicator bit* which indicates whether or not the agent is a *source*. Each source holds a binary *input bit* and each agent (including sources) holds a binary *opinion*. . Note that having the indicator bit equal to 1 is equivalent to possessing an input bit: both are exclusive properties of source nodes. However, we keep them distinct for a clearer presentation. The number of sources (i.e., agents whose indicator bit is 1) is denoted by k. We denote by k_0 and k_1 the number of sources whose input bit is initially set to 1 and 0, respectively. Assuming $k_1 \neq k_0$, we define the *majority bit*, termed b_{maj}, as 1 if $k_1 > k_0$ and 0 if $k_1 < k_0$. Source agents know that they are sources (using the indicator bit) but they do not know whether they hold the majority bit. The parameters k, k_1 or k_0 are not known to the sources or to any other agent. It is required that the opinions of all agents eventually converge to the majority bit b_{maj}.

We note that agents hold their output and indicator bits privately, and we do not require them to necessarily reveal these bits publicly (in their visible parts) unless they wish to. To avoid dealing with the cases where the number of sources holding the majority bit is arbitrarily close to $\frac{k}{2}$, we shall guarantee correctness (w.h.p.) only if the fraction of sources holding the majority is bounded away from $\frac{1}{2}$, i.e., only if $|\frac{k_1}{k_0} - 1| > \epsilon$, for some positive constant ϵ. When $k = 1$, the problem is called *Bit Dissemination*, for short. Note that in this case, the single source agent holds the bit b_{maj} to be disseminated and there is no other source agent introducing a conflicting opinion.

T-Clock Synchronization. Let T be an integer. In the *T-Clock Synchronization* problem, each agent maintains a *clock* modulo T that is incremented at each round. The goal of agents is to converge on having the same value in their clocks modulo T. (We may omit the parameter T when it is clear from the context.)

Probabilistic self-stabilization and convergence. Self-stabilizing protocols are meant to guarantee that the system eventually converges to a *legal* configuration regardless of the initial states of the agents [22]. Here we use a slightly weaker notion, called *probabilistic self-stabilization*, where stability is guaranteed w.h.p. [10]. More formally, for the Clock Synchronization problem, we assume that *all*

states are initially set by an adversary. For the Majority Bit Dissemination problem, we assume that *all* states are initially set by an adversary except that it is assumed that the agents know their total number n, and that this information is not corrupted.

In the context of T-Clock Synchronization, a legal configuration is reached when all clocks show the same time modulo T, and in the Majority Bit Dissemination problem, a legal configuration is reached when all agents output the majority bit b_{maj}. Note that in the context of the Majority Bit Dissemination problem, the legality criterion depends on the initial configuration (that may be set by an adversary). That is, the agents must converge their opinion on the majority of input bits of sources, as evident in the initial configuration.

The system is said to *stabilize* in t rounds if, from any initial configuration w.h.p., within t rounds it reaches a legal configuration and remains legal for at least some polynomial time [10, 12, 24]. In fact, for the self-stabilizing Bit Dissemination problem, if there are no conflicting source agents holding a minority opinion (such as in the case of a single source agent), then our protocols guarantee that once a legal configuration is reached, it remains legal indefinitely. Note that, for any of the problems, we do not require that each agent irrevocably commits to a final opinion but that eventually agents arrive at a legal configuration without necessarily being aware of that.

1.4 Our Results Our main results are the following.

Theorem 1.1. *Fix an arbitrarily small constant $\epsilon > 0$. There exists a protocol, called* Syn-Phase-Spread, *which solves the Majority Bit Dissemination problem in a self-stabilizing manner in $\tilde{\mathcal{O}}(\log n)$ rounds[2] w.h.p using 3-bit messages, provided that the majority bit is supported by at least a fraction $\frac{1}{2} + \epsilon$ of the source agents.*

Theorem 1.1 is proved in Section 5. The core ingredient of Syn-Phase-Spread is our construction of an efficient self-stabilizing T-Clock Synchronization protocol, which is used as a black-box. For this purpose, the case that interests us is when $T = \tilde{\mathcal{O}}(\log n)$. Note that in this case, the following theorem, proved in Section 4, states that the convergence time of the Clock Synchronization algorithm is $\tilde{\mathcal{O}}(\log n)$.

Theorem 1.2. *Let T be an integer. There exists a self-stabilizing T-Clock Synchronization protocol, called* Syn-Clock, *which employs only 3-bit messages, and synchronizes clocks modulo T within $\tilde{\mathcal{O}}(\log n \log T)$ rounds w.h.p.*

In addition to the self-stabilizing context our protocols can tolerate the presence of Byzantine agents, as long as their number is[3] $\mathcal{O}(n^{1/2-\epsilon})$. However, in order to focus on the self-stabilizing aspect of our results, in this work we do not explicitly address the presence of Byzantine agents.

The proofs of both Theorem 1.2 and Theorem 1.1 rely on recursively applying a new general compiler which can essentially transform any self-stabilizing algorithm with a certain property (called "the bitwise-independence property") that uses ℓ-bit messages to one that uses only $\lceil \log \ell \rceil + 1$-bit messages, while paying only a small penalty in the running time. This compiler is described in Section 3, in Theorem 3.1, which is also referred as "the Message Reduction Theorem". The structure between our different lemmas and results is summarized in the picture below, Figure 2.

[2] With a slight abuse of notation, with $\tilde{\mathcal{O}}(f(n)g(T))$ we refer to $f(n)g(T)\log^{\mathcal{O}(1)}(f(n))\log^{\mathcal{O}(1)}(g(T))$. All logarithms are in base 2.

[3] Specifically, it is possible to show that, as a corollary of our analysis and the fault-tolerance property of the analysis in [24], if $T \leq poly(n)$ then Syn-Clock can tolerate the presence of up to $\mathcal{O}(n^{1/2-\epsilon})$ Byzantine agents for any $\epsilon > 0$. In addition, Syn-Phase-Spread can tolerate $\min\{(1-\epsilon)(k_{maj} - k_{min}), n^{1/2-\epsilon}\}$ Byzantine agents, where k_{maj} and k_{min} are the number of sources supporting the majority and minority opinions, respectively. Note that for the case of a single source ($k = 1$), no Byzantine agents are allowed; indeed, a single Byzantine agent pretending to be the source with the opposite opinion can clearly ruin any protocol.

It remains an open problem, both for the self-stabilizing Bit Dissemination problem and for the self-stabilizing Clock Synchronization problem, whether the message size can be reduced to 2 bits or even to 1 bit, while keeping the running time poly-logarithmic.

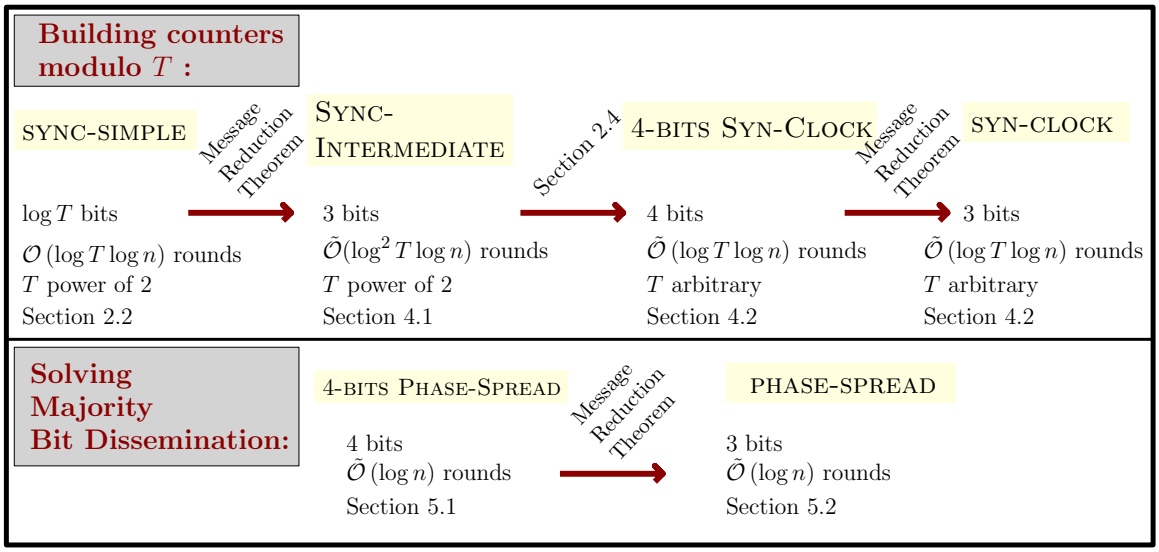

Figure 2: The structure of our arguments. Note that the Message Reduction Theorem is used on three occasions.

1.5 Related work The computational study of abstract systems composed of simple individuals that interact using highly restricted and stochastic interactions has recently been gaining considerable attention in the community of theoretical computer science. Popular models include *population protocols* [3, 7, 5, 9], which typically consider constant size individuals that interact in pairs (using constant size messages) in random communication patterns, and the *beeping* model [1, 32], which assumes a fixed network with extremely restricted communication. Our model also falls in this framework as we consider the $\mathcal{PULL}$ model [21, 38, 39] with constant size messages. So far, despite interesting works that consider different fault-tolerant contexts [4, 5, 9], most of the progress in this framework considered non-faulty scenarios.

Information dissemination is one of the most well-studied topics in the community of distributed computing, see, *e.g.*, [4, 16, 21, 23, 24, 33, 38]. Classical examples include the *Broadcast* (also referred to in the literature as *Rumor Spreading*) problem, in which a piece of information residing at one source agent is to be disseminated to the rest of the population, and *majority-consensus* (here, called Majority Bit Dissemination) problems in which processors are required to agree on a common output value which is the majority initial input value among all agents [4, 40] or among a set of designated source agents [33]. An extensive amount of research has been dedicated to study such problems in $\mathcal{PUSH}/\mathcal{PULL}$ based protocols (including the *phone call* model), due to the inherent simplicity and fault-tolerant resilience of such meeting patterns. Indeed, the robustness of $\mathcal{PUSH}/\mathcal{PULL}$ based protocols to weak types of faults, such as crashes of messages and/or agents, or to the presence of relatively few Byzantine agents, has been known for quite a while [31, 38]. Recently, it has been shown that under the $\mathcal{PUSH}$ model, there exist efficient Broadcast and Majority Bit Dissemination protocols that use a single bit per message and can overcome flips in messages (noise) [33]. The protocols therein,

however, heavily rely on the assumption that agents know when the protocol has started. Observe that in a self-stabilizing context, in which the adversary can corrupt the initial clocks setting them to arbitrary times, such an assumption would be difficult to remove while preserving the small message size.

In general, there are only few known self-stabilizing protocols that operate efficiently under stochastic and capacity restricted interactions. An example, which is also of high relevance to this paper, is the work of Doerr et al. on *Stabilizing Consensus* [24] operating in the $\mathcal{PULL}$ model. In that work, each agent initially has a state taken out of a set of m opinions and the goal is to converge on one of the proposed states. The proposed algorithm which runs in logarithmic time is based on sampling the states of 2 agents and updating the agent's state to be the median of the 2 sampled states and the current state of the agent (3 opinions in total). Since the total number of possible states is m, the number of bits that must be revealed in each interaction is $\Omega(\log m)$. Another example is the plurality consensus protocol in [12], in which each agent has initially an opinion and we want the system to converge to the most frequent one in the initial configuration of the system. In fact, the Majority Bit Dissemination problem can be viewed as a generalization of the *majority-consensus* problem (i.e. the plurality consensus problem with two opinions), to the case in which multiple agents may initially be unopinionated. In the previous sense, we also contribute to the line of research on the majority-consensus problem [11, 19, 30].

Another fundamental building block is Clock Synchronization [8, 41, 42, 43]. We consider a synchronous system in which clocks tick at the same pace but may not share the same opinion. This version has earlier been studied in e.g., [13, 25, 27, 28, 34, 37] under different names, including "digital Clock Synchronization" and "synchronization of phase-clocks"; We simply use the term "Clock Synchronization". There is by now a substantial line of work on Clock Synchronization problems in a self-stabilizing context [26, 28, 45, 44]. We note that in these papers the main focus is on the resilience to Byzantine agents. The number of rounds and message lengths are also minimized, but typically as a function of the number of Byzantine processors. Our focus is instead on minimizing the time and message complexities as much as possible. The authors in [45, 44] consider mostly a deterministic setting. The communication model is very different than ours, as every agent gets one message from every other agent on each round. Moreover, agents are assumed to have unique identifiers. In contrast, we work in a more restricted, yet randomized communication setting. In [26, 45] randomized protocols are also investigated. We remark that the first protocol we discuss Syn-Simple (Proposition 2.1), which relies on a known simple connection between consensus and counting [26], already improves exponentially on the randomized algorithms from [26, 45] in terms of number of rounds, number of memory states, message length and total amount of communication, in the restricted regime where the resilience parameter f satisfies $\log n \leq f \leq \sqrt{n}$. We further note that the works [44, 45] also use a recursive construction for their clocks (although very different from the one we use in the proof of Theorem 1.2). The induction in [45] is on the resilience parameter f, the number of agents and the clock length together. This idea is improved in [44] to achieve optimality in terms of resilience to Byzantine agents.

To the best of our knowledge there are no previous works on self-stabilizing Clock Synchronization or Majority Bit Dissemination that aim to minimize the message size beyond logarithmic in the $\mathcal{PULL}$ model.

2 Preliminaries

2.1 A majority based, self-stabilizing protocol for consensus on one bit Let us recall[4] the stabilizing consensus protocol by Doerr et al. in [24]. In this protocol, called MAJ-CONSENSUS, each agent holds an opinion. In each round each agent looks at the opinions of two other random agents and updates her opinion taking the majority among the bits of the observed agents and its own. Note that this protocol uses only a single bit per interaction, namely, the opinion. The usefulness of MAJ-CONSENSUS comes from its extremely fast and fault-tolerant convergence toward an agreement among agents, as given by the following result.

THEOREM 2.1. (DOERR ET AL. [24]) *From any initial configuration,* MAJ-CONSENSUS *converges to a state in which all agents agree on the same output bit in $\mathcal{O}(\log n)$ rounds, w.h.p. Moreover, if there are at most $\kappa \leq n^{1/2-\epsilon}$ Byzantine agents, for any constant $\epsilon > 0$, then after $\mathcal{O}(\log n)$ rounds all non-Byzantine agents have converged and consensus is maintained for $n^{\Omega(1)}$ rounds w.h.p.*[5]

2.2 Protocol SYN-SIMPLE: A simple protocol with many bits per interaction We now present a simple self-stabilizing T-Clock Synchronization protocol, called SYN-SIMPLE, that uses relatively many bits per message, and relies on the assumption that T is a power of 2. The protocol is based on iteratively applying a self-stabilizing consensus protocol on each bit of the clock separately, and in parallel.

Formally, each agent u maintains a clock $C_u \in [0, T-1]$. At each round, u displays the opinion of her clock C_u, pulls 2 uniform other such clock opinions, and updates her clock as the bitwise majority of the two clocks it pulled, and her own. Subsequently, the clock C_u is incremented. We present the pseudo code of SYN-SIMPLE in Algorithm 1.

SYN-SIMPLE protocol

1 u samples two agents u_1 and u_2.

2 u updates its clock with the bitwise majority of its clock and those of the sample nodes.

3 u increments its clock by one unit.

Algorithm 1: One round of SYN-SIMPLE, executed by each agent u.

We prove the correctness of SYN-SIMPLE in the next proposition.

PROPOSITION 2.1. *Let T be a power of 2. The protocol* SYN-SIMPLE *is a self-stabilizing protocol that uses $\mathcal{O}(\log T)$ bits per interaction and synchronizes clocks modulo T in $\mathcal{O}(\log T \log n)$ rounds w.h.p.*

Proof. Let us look at the least significant bit. One round of SYN-SIMPLE is equivalent to one round of MAJ-CONSENSUS with an extra flipping of the opinion due to the increment of the clock. The crucial

[4] Our protocols will use this protocol as a *black box*. However, we note that the constructions we outline are in fact independent of the choice of consensus protocol, and this protocol could be replaced by other protocols that achieve similar guarantees.

[5] The original statement of [24] says that if at most $\kappa \leq \sqrt{n}$ agents can be corrupted at any round, then convergence happens for all but at most $\mathcal{O}(\kappa)$ agents. Let us explain how this implies the statement we gave, namely that we can replace $\mathcal{O}(\kappa)$ by κ, if $\kappa \leq n^{\frac{1}{2}-\epsilon}$. Assume that we are in the regime $\kappa \leq n^{\frac{1}{2}-\epsilon}$. It follows from [24] that all but a set of $\mathcal{O}(\kappa)$ agents reach consensus after $\mathcal{O}(\log n)$ round. This set of size $\mathcal{O}(\kappa)$ contains both Byzantine and non Byzantine agents. However, if the number of agents holding the minority opinion is $\mathcal{O}(\kappa) = \mathcal{O}(n^{1/2-\epsilon})$, then the expected number of non Byzantine agents that disagree with the majority *at the next round* is in expectation $\mathcal{O}(\kappa^2/n) = \mathcal{O}(n^{-2\epsilon})$. Thus, by Markov's inequality, this implies, that at the next round consensus is reached among *all non-Byzantine agents* w.h.p. Note also that, for the same reasons, the Byzantine agents do not affect any other non-Byzantine agent for n^{ϵ} rounds w.h.p.

point is that all agents jointly flip their bit on every round. Because the function agents apply, MAJ, is symmetric, it commutes with the flipping operation. More formally, let $\vec{b}_t$ be the vector of the first bits of the clocks of the agents at round t under an execution of SYN-SIMPLE. E.g. $(\vec{b}_t)_u$ is the value of the less significant bit of node u's clock at time t. Similarly, we denote by $\vec{c}_t$ the first bits of the clocks of the agents at round t obtained by running a modified version of SYN-SIMPLE in which *time is not incremented* (i.e. we skip line 3 in Algorithm 1). We couple $\vec{b}$ and $\vec{c}$ trivially, by running the two versions on the same interaction pattern (in other words, each agent starts with the same memory and pulls the same agents at each round in both executions). Then, $\vec{b}_t$ is equal to $\vec{c}_t$ when t is even, while is equal to $\vec{b}_t = \mathbf{1} - \vec{c}_t$ when t is odd. Moreover, we know from Theorem 2.1 that $\vec{c}_t$ converge to a stable opinion in a self-stabilizing manner. It follows that, from any initial configuration of states (i.e. clocks), w.h.p, after $\mathcal{O}(\log n)$ rounds of executing SYN-SIMPLE, all agents share the same opinion for their first bit, and jointly flip it in each round. Once agents agree on the first bit, since T is a power of 2, the increment of time makes them flip the second bit *jointly* once every 2 rounds. More generally, assuming agents agree on the first ℓ bits of their clocks, they *jointly* flip the $\ell+1$'st bit once every 2^ℓ rounds, on top of doing the MAJ-CONSENSUS protocol on that bit. Hence, the same coupling argument shows that the flipping doesn't affect the convergence on bit $\ell+1$. Therefore, $\mathcal{O}(\log n)$ rounds after the first ℓ bits are synchronized, w.h.p. the $\ell+1$'st bit is synchronized as well. The result thus follows by induction.

2.3 The bitwise-independence property Our general transformer described in Section 3 is useful for reducing the message size of protocols with a certain property called bitwise-independence. Before defining the property we need to define a variant of the $\mathcal{PULL}$ model, which we refer to as the $\mathcal{BIT}$ model. The reason we introduce such a variant is mainly technical, as it appears naturally in our proofs. Thus, unless explicitly stated, we always refer to the $\mathcal{PULL}$ model.

Recall that in the $\mathcal{PULL}(\eta,\ell)$ model, at any given round, each agent u is reading an ℓ-bit message m_{v_j} for each of the η observed agents v_j chosen u.a.r. (in our case $\eta = 2$), and then, in turn, u updates her state according to the instructions of a protocol P. Informally, in the $\mathcal{BIT}$ model, each agent u also receives η messages, however, in contrast to the $\mathcal{PULL}$ model where each such message corresponds to one observed agent, in the $\mathcal{BIT}$ model, the i'th bit of each such message is received independently from a (typically new) agent, chosen u.a.r. from all agents.

DEFINITION 1. (THE $\mathcal{BIT}$ MODEL) *In the $\mathcal{BIT}$ model, at each round, each agent u picks $\eta\ell$ agents u.a.r., namely, $v_1^{(1)}, v_2^{(1)}, \ldots v_\ell^{(1)}, \ldots, v_1^{(\eta)}, v_2^{(\eta)}, \ldots v_\ell^{(\eta)}$, and reads $\hat{s_i}^{(j)} = s_i(v_i^{(j)})$, the i-th bit of the visible part of agent $v_i^{(j)}$, for every $i \le \ell$ and $j \le \eta$. For each $j \le \eta$, let $\hat{m}_j(u)$ be the ℓ-bit string $\hat{m}_j(u) := (\hat{s_1}^{(j)}, \hat{s_2}^{(j)}, \ldots, \hat{s_\ell}^{(j)})$. By a slight abuse of language we call the strings $\{\hat{m}_j(u)\}_{j\le\eta}$ the* messages *received by u in the $\mathcal{BIT}$ model.*

DEFINITION 2. (THE *bitwise – independence* PROPERTY) *Consider a protocol* P *designed to work in the $\mathcal{PULL}$ model. We say that* P *has the bitwise-independence property if its correctness and running time guarantees remain the same under the $\mathcal{BIT}$ model (assuming that given the messages $\{\hat{m}_j(u)\}_{j\le\eta}$ it receives at any round, each agent u performs the same actions that it would have, had it received these messages in the $\mathcal{PULL}$ model).*

Let us first state a fact about protocols having the bitwise-independence property.

LEMMA 2.1. *Assume we are given two protocols* SYN-GENERIC *and* P, *designed to work in the $\mathcal{PULL}$ model, such that*

- *Protocol* SYN-GENERIC *synchronizes clocks modulo T for some T and*

- *Protocol* P *works assuming agents share a clock modulo* T.

Denote by Syn-P *the parallel execution of* Syn-Generic *and* P, *with* P *using the clock synchronized by* Syn-Generic. *Then*

1. *If* Syn-Generic *and* P *are self-stabilizing then so is* Syn-P, *and the convergence time of* Syn-P *is at most the sum of convergence times of* Syn-Generic *and* P.

2. *Finally, if* Syn-Generic *and* P *have the bitwise-independence property, and* P *is also self-stabilizing,* Syn-P *has the bitwise-independence property too.*

Proof. The self-stabilizing property of Syn-P and its convergence time directly follow from those of Syn-Generic and P (part 1 of the statement). We just need to check the correctness of Syn-P, when run in the $\mathcal{BIT}$ model (part 2 of the statement). The fact that Syn-Generic and P are run in parallel means that the part of the message and computations regarding Syn-Generic are not affected by those regarding P. This still holds when running the protocol in the $\mathcal{BIT}$ model. Since, by hypothesis, Syn-Generic has the independence property, there exists a time τ after which all agents share a synchronized clock modulo T, even in the $\mathcal{BIT}$ model. Thus, after time τ, we can disregard the part of the message corresponding to Syn-Generic, and view the execution of Syn-P as simply P. The assumption that P is self-stabilizing and has the independence property implies that, regardless of the nodes' memory states concerning the execution of P at time τ, Syn-P still works in the $\mathcal{BIT}$ model as in the original $\mathcal{PULL}$ model, thus inheriting the bitwise-independence property from Syn-Generic and P.

We next show that the protocol Syn-Simple has the aforementioned bitwise-independence property.

Lemma 2.2. Syn-Simple *has the bitwise-independence property.*

Proof. Let us start by commenting on Syn-Simple, when run in the usual $\mathcal{PULL}$ model. Let ℓ' be the size of the clocks. Assume the first $i < \ell'$ bits of the clocks have been synchronized. At this stage, the $(i+1)$-st bit of each agent u is flipped every 2^i rounds (from 0 to 1 or from 1 to 0) and updated as the majority of the $(i+1)$-st bit of $C(u)$ and the 2 pulled messages on each round. Since the first i bits are synchronized, the previous flipping is performed by all agents at the same round. Let us now consider the protocol over the $\mathcal{BIT}$ model. Observe that, in order for Syn-Simple to work, we do not need the bit at index $(i+1)$ to come from the same agent as the bits corresponding to indices $\leq i$, as long as convergence on the first i bits has been achieved. Hence, as is, the reasoning above for the $\mathcal{PULL}$ model holds in the $\mathcal{BIT}$ model as well.

3 A General Compiler that Reduces Message Size

In this section we present a general compiler that allows to implement a protocol P using ℓ-bit messages while using messages of order $\log \ell$ instead, as long as P enjoys the bitwise-independence property. The compiler is based on replacing a message by an index to a given bit of the message. This tool will repeatedly be used in the following sections to obtain our Clock Synchronization and Majority Bit Dissemination algorithms that use 3-bit messages.

Theorem 3.1. (the Message Reduction Theorem) *Any self-stabilizing protocol* P *in the* $\mathcal{PULL}(\eta, \ell)$ *model having the bitwise-independence property, and whose running time is* L_{P}, *can be emulated by a protocol* Emul(P) *which runs in the* $\mathcal{PULL}(2, \lceil \log(\frac{\eta}{2}\ell) \rceil + 1)$ *model, has running time* $\mathcal{O}(\log(\eta\ell)\log n) + \frac{\eta}{2}\ell L_{\mathrm{P}}$ *and has itself the bitwise-independence property.*

Remark 1. *The only reason for designing* Emul(P) *to run in the* $\mathcal{PULL}(2,\lceil\log(\frac{\eta}{2}\ell)\rceil+1)$ *model in the Message Reduction Theorem is the consensus protocol we adopt,* maj-consensus, *which works in the* $\mathcal{PULL}(2)$ *model. In fact,* Emul(P) *can be adapted to run in the* $\mathcal{PULL}(1,\lceil\log(\eta\ell)\rceil+1)$ *model by using a consensus protocol which works in the* $\mathcal{PULL}(1)$ *model. However, no self-stabilizing binary consensus protocol in the* $\mathcal{PULL}(1)$ *model with the same performances as* maj-consensus *is currently known.*

Proof of Theorem 3.1. Let $s(u) \in \{0,1\}^\ell$ be the message displayed by an agent u under P at a given round. For simplicity's sake, in the following we assume that η is even, the other case is handled similarly. In Emul(P), agent u keeps the message $s(u)$ privately, and instead displays a clock $C(u)$ written on $\lceil\log(\frac{\eta}{2}\ell)\rceil$ bits, and one bit of the message $s(u)$, which we refer to as the P-bit. Thus, the total number of bits displayed by the agent operating in Emul(P) is $\lceil\log(\frac{\eta}{2}\ell)\rceil+1$. The purpose of the clock $C(u)$ is to indicate to agent u which bit of $s(u)$ to display. In particular, if the counter has value 0, then the 0-th bit (i.e the least significant bit) of $s(u)$ is shown as the P-bit, and so on. In what follows, we refer to $s(u)$ as the *private message* of u, to emphasize the fact that this message is not visible in Emul(P). See Figure 3 for an illustration.

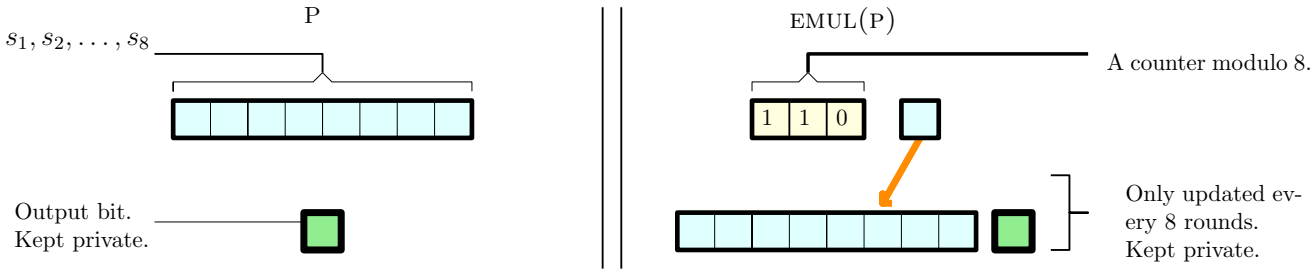

Figure 3: On the left is a protocol P using $\ell = 8$ bits in total and pulling only one node per round ($\eta = 1$). On the right is the emulated version Emul(P) which uses 4 bits only. The bits depicted on the bottom of each panel are kept privately, while the bits on the top are public, that is, appear in the visible part.

Each round of P executed in the $\mathcal{PULL}(\eta,\ell)$ model by an agent u is emulated by $\frac{\eta}{2}\ell$ rounds of Emul(P) in the $\mathcal{PULL}(2,\lceil\log(\frac{\eta}{2}\ell)\rceil+1)$ model. We refer to such $\frac{\eta}{2}\ell$ rounds as a *phase*, which is further divided to $\frac{\eta}{2}$ subphases of length ℓ. Note that since each agent samples 2 agents in a round, the total number of agents sampled by an agent during a phases is $\eta\ell$.

For a generic agent u, a phase starts when its clock $C(u)$ is zero, and ends after a full loop of its clock (i.e. when $C(u)$ returns to zero). Each agent u is running protocol Syn-Simple on the $\lceil\log(\frac{\eta}{2}\ell)\rceil$ bits which correspond to her clock $C(u)$. Note that the phases executed by different agents may initially be unsynchronized, but, thanks to Proposition 2.1, the clocks $C(u)$ eventually converge to the same value, for each agent u, and hence all agents eventually agree on when each phase (and subphase) starts.

Let u be an arbitrary agent. Denote by $\hat{s}_1^{(1)}, \hat{s}_2^{(1)}, \ldots \hat{s}_\ell^{(1)}$, ..., $\hat{s}_1^{(\eta)}, \hat{s}_2^{(\eta)}, \ldots \hat{s}_\ell^{(\eta)}$ the P-bits collected by u from agents chosen u.a.r during a phase. Consider a phase and a round $z \in \{1, \cdots, \frac{\eta}{2}\ell\}$ in that phase. Let i and j be such that $z = j \cdot \ell + i$. We view z as round i of subphase $j+1$ of the phase. On this round, agent u pulls two messages from agents v and w, chosen u.a.r. Once the clocks (and thus phases and subphases) have synchronized, agents v and w are guaranteed to be displaying the ith index of their private messages, namely, the values $s_i(v)$ and $s_i(w)$, respectively. Agent u then sets $\hat{s}_i^{(2j-1)}$ equal to $s_i(v)$ and $\hat{s}_i^{(2j)}$ equal to $s_i(w)$.

In Emul(P), the messages displayed by agents are only updated after a full loop of C. It therefore follows from the previous paragraph that the P-bits collected by agent u after a full-phase are

distributed like the bits collected during one round of P in the $\mathcal{BIT}$ model (see Definition 1), assuming the clocks are synchronized already.

Correctness. The bitwise-independence property of Syn-Simple (Lemma 2.2), implies that Syn-Simple still works when messages are constructed from the P-bits collected by Emul(P). Therefore, from Proposition 2.1, eventually all the clocks C are synchronized. Since private messages s are only updated after a full loop of C, once the clocks C are synchronized a phase of Emul(P) corresponds to *one* round of P, executed in the $\mathcal{BIT}$ model. Hence, the hypothesis that P operates correctly in a self-stabilizing way in the $\mathcal{BIT}$ model implies the correctness of Emul(P).

Running time. Once the clocks $C(u)$ are synchronized, for all agents u, using the first $\lceil \log(\frac{\eta}{2}\ell) \rceil$ bits of the messages, the agents reproduce an execution of P with a multiplicative time-overhead of $\frac{\eta}{2}\ell$. Moreover, from Proposition 2.1, synchronizing the clocks $C(u)$ takes $\mathcal{O}(\log(\eta m)\log n)$ rounds. Thus, the time to synchronize the clocks costs only an additive factor of $\mathcal{O}(\log(\eta m)\log n)$ rounds, and the total running time is $\mathcal{O}(\log(\eta m)\log n) + \frac{\eta}{2}\ell \cdot L_{\mathrm{P}}$.

Bitwise-independence property. Protocol Emul(P) inherits the bitwise-independence property from that of Syn-Simple (Lemma 2.2) and P (which has the property by hypothesis): We can apply Lemma 2.1 where Syn-Generic is Syn-Simple and P is the subroutine described above, which displays at each round the bit of P whose index is given by a synchronized clock C modulo ℓ (i.e. the one produced by Syn-Simple). Observe that the aforementioned subroutine is self-stabilizing, since it emulates P once clocks are synchronized. Then, in the notation of Lemma 2.1, Emul(P) is Syn-P. □

4 Self-Stabilizing Clock Synchronization

In Section 2.2 we described Syn-Simple - a simple self-stabilizing Clock Synchronization protocol that uses $\log T$ bits per interaction. In this section we describe our main self-stabilizing Clock Synchronization protocol, Syn-3Bits, that uses only 3 bits per interaction. We first assume T is a power of 2. We show how to get rid of this assumption in Section 4.2.

4.1 Clock Synchronization with 3-bit messages, assuming T is a power of two

In this section, we show the following result.

Lemma 4.1. *Let T be a power of 2. There exists a synchronization protocol* Syn-Intermediate *which synchronizes clocks modulo T in time $\tilde{\mathcal{O}}\left(\log^2 T \log n\right)$ using only 3-bit messages. Moreover,* Syn-Intermediate *has the bitwise-independence property.*

Before presenting the proof of Lemma 4.1, we need a remark about clocks.

Remark 2. *In order to synchronize a clock C modulo T, throughout the analysis we often obtain a clock C' modulo T* which is incremented every ℓ rounds*. However, C' can still be translated back to a clock modulo T which is incremented every round, by keeping a third clock C'' modulo ℓ and setting*

$$C = \ell C' + C'' \mod T.$$

Proof of Lemma 4.1. At a high level, we simply apply iteratively the Message Reduction Theorem in order to reduce the message to 3 bits, starting with P = Syn-Simple. A pictorial representation of our recursive protocol is given in Figure 4, and a pseudocode is given in Algorithm 2. The pseudocode deviates slightly from the presentation done in the proof, as it makes no use of recursion.

Let us consider what we obtain after applying the Message Reduction Theorem the first time to P =Syn-Simple for clocks modulo T. Recall that we assume that T is a power of 2. From Proposition 2.1 we know that in this case, the convergence time of Syn-Simple is $L_{\mathrm{P}} = \mathcal{O}(\log T \log n)$, the number of pulled agents at each round is 2 and the number of bits of each message is $\ell = \log T$.

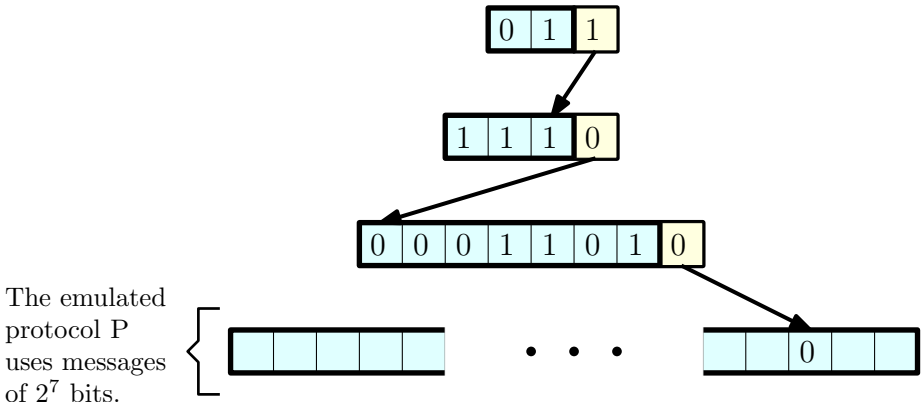

Figure 4: A more explicit view of our 3-bit emulation of protocol P, obtained by iterating Lemma 3.1. The down-most layer represents the 2^7-bits message displayed by protocol P. Each layer on the picture may be seen as the message of a protocol emulating P with fewer bits, that is, as we go up on the figure we obtain more and more economical protocols in terms of message length. In particular, the top layer represents the 3-bit message in the final emulation. The left-most part of each message (colored in light blue) encodes a clock. The right-most bit (colored in light yellow) of each message (except the bottom-most one) corresponds to a particular bit of the layer *below* it. The index of this particular displayed bit is given by the value of the clock. Each clock on an intermediate layer is updated only when the clock on the layer *above* completes a loop (i.e., has value 0). The clock on the top-most layer is updated on every round.

With the emulation produced by the Message Reduction Theorem, the clock used in P =Syn-Simple is incremented only every $\ell = \log T$ rounds. Another way to interpret this is that we obtain a clock modulo $T \cdot \ell$ and using Remark 2 we can turn it into a counter modulo T that is incremented at each round. Hence, by the running time analysis of the Message Reduction Theorem, we obtain a protocol Emul(P) which synchronizes a clock modulo T in $\mathcal{O}(\log n \log\log T) + \mathcal{O}\left(\log^2 T \log n\right) = \mathcal{O}\left(\log^2 T \log n\right)$ rounds. The message size is reduced from $\log T$ to $\lceil \log\log T \rceil + 1 = \mathcal{O}(\log\log T)$.

By repeatedly applying the Message Reduction Theorem, we reduce the size of the message ℓ as long as $\ell > \lceil \log \ell \rceil + 1$, i.e. as long as $\ell > 3$. The number of repeated application of the Message Reduction Theorem until the message size is 3 is thus of order $\log^* T$.

Let us analyze the running time. Let $\ell_1 = \log T$, $\ell_{i+1} = \lceil \log \ell_i \rceil + 1$ and let $\tau(T) = \tau$ be the smallest integer such that $\ell_\tau = 3$. We apply the Message Reduction Theorem $i \leq \tau$ times, and we obtain a message size ℓ_i and a running time L_i, such that

$$L_{i+1} \leq \gamma_1(\log \ell_i \log n + \ell_i L_i), \tag{4.1}$$

for some constant γ_1 independent of i. Let a and b be two given numbers. Given two real numbers a and b, we use the notation $a \vee b$ to denote the maximum of a and b. Set L_1 to be $L_1 := L_{\text{Syn-Simple}} \vee \log n = \mathcal{O}(\log T \log n) \vee \log n$, taking the maximum with $\log n$ for technical convenience. The second term dominates in Equation (4.1) because $\ell_i >> \log \ell_i$ and $L_i > \log n$. Hence, we obtain from Equation (4.1) that $L_{i+1} \leq 2\gamma_1 \ell_i L_i$. It follows by induction that

$$L_{i+1} \leq (2\gamma_1)^i L_1 \prod_{j=1}^{i} \ell_j.$$

Syn-Intermediate protocol

Memory: Each agent u keeps a sequence of clocks $C_1, \dots, C_\tau$ and a sequence of bits $b_1, \dots, b_\tau$. The clock C_1 runs modulo T, the clock C_τ runs modulo 4, and the i-th clock C_i runs modulo $2^{\ell_i - 1}$ (see proof of Lemma 4.1). Each agent u also maintains a sequence of heaps (or some ordered structure) S_i^δ, for each $\delta \in \{1, 2\}$ and $i = 1, \dots, \tau$.

Message: u displays C_τ (2 bits) and b_τ (1 bit). For all $i \in [\tau], b_i(u)$ is the $C_i(u)$-th bit of the string obtained concatenating the binary representation of $C_{i-1}(u)$ and $b_{i-1}(u)$.

1 u samples two agents u_1 and u_2.

2 u updates its clock with the bitwise majority of its clock and those of the sampled nodes.

3 u increments its clock by one unit.

4 u sets i^* equal to the maximal $i < \tau$ such that $C_{i+1} \neq 0$.

5 For $\delta = 1, 2$, u pushes $b_\tau(u_\delta)$ in $S_{i^*}^\delta$.
(Note that, if $C_{i^*+1}, \dots, C_\tau$ are synchronized, then all agents are displaying the bit with index C_{i^*+1} of (C_{i^*}, b_{i^*}) as b_τ.)

6 While $i > 1$ and $C_i = 0$, u does the following:

7 | Pops the last $\ell_{i-1} - 1$ bits from S_{i-1}^δ and set s^δ equal to it.

8 | Sets C_{i-1} equal to the bitwise majority of $C_{i-1}(u)$, s^1 and s^2.

9 | Increments C_{i-1} and decrement i by one unit.

Algorithm 2: Iterative version of the protocol Syn-Intermediate, executed by each agent u, unfolding the recursion in proof of Lemma 4.1.

The running time of Emul(P) =Syn-Clock after the last application of the Message Reduction Theorem, i.e. τ, is thus

$$L_{\text{Syn-Clock}} := L_\tau \leq (2\gamma_1)^\tau L_1 \prod_{i=1}^{\tau-1} \ell_i.$$

We use the following fact.

Fact 4.1. *If $|x| < 1$, it holds*

$$e^{\frac{x}{1+x}} \leq 1 + x \leq e^x \leq 1 + \frac{x}{1-x}.$$

From the bounds $L_1 = \mathcal{O}(\log T \log n)$, $\prod_{i=1}^{\tau} \ell_i \leq \ell_1 \ell_2 \ell_3^\tau$, $\ell_1 = \mathcal{O}(\log T)$, $\ell_2 = \mathcal{O}(\log\log T)$ and Lemma B.1, we obtain $(2\gamma_1)^\tau = (\log\log\log T)^{\mathcal{O}(1)} = \mathcal{O}(\log\log T)$ and

$$\ell_3^\tau \leq 2^{\mathcal{O}\left((\log\log\log\log T)^2\right)} \leq 2^{\mathcal{O}(\log\log\log T)} \leq (\log\log T)^{\mathcal{O}(1)}.$$

We thus conclude that

$$L_{\text{Syn-Clock}} \leq (2\gamma_1)^\tau \prod_{i=1}^{\tau} \ell_i L_1 \leq \mathcal{O}(\log\log T) \cdot \ell_1 \ell_2 \ell_3^\tau \cdot \mathcal{O}(\log T \log n)$$

$$\leq \mathcal{O}(\log\log T)\cdot\mathcal{O}(\log T)\cdot\mathcal{O}(\log\log T)\cdot\mathcal{O}(\log\log T)^{\mathcal{O}(1)}\cdot\mathcal{O}(\log T\log n)$$
$$\leq \log^2 T\log n\cdot(\log\log T)^{\mathcal{O}(1)}.$$

The total slowdown with respect to Syn-Simple corresponds to $\prod_{i=1}^{\tau}\ell_i = \tilde{\mathcal{O}}(\log T)$. Hence the clock produced by the emulation is incremented every $\tilde{\mathcal{O}}(\log T)$ rounds. In other words we obtain a clock modulo $T\cdot f(T)$ for some function f. But using Remark 2 we can still view this as a clock modulo T. □

4.2 Extension to general T and running time improvement. In this subsection we aim to get rid of the assumption that T is a power of 2 in Lemma 4.1, and also reduce the running time of our protocol to $\tilde{\mathcal{O}}(\log n\log T)$, proving Theorem 1.2.

Syn-Clock protocol

Memory: Each agent u stores a clock $C'(u)$ which runs modulo $T' \gg \gamma\log n\log T$. Each agent u also stores a variable Q which is incremented only once every T' rounds and runs modulo T.

Message: Each agent u displays 4 bits. On the first 3 bits, protocol Syn-Intermediate is applied to synchronize C'. The 4-th bit $b(u)$ is the bit with index $(\lfloor\frac{C'(u)}{\gamma\log n}\rfloor \mod \lceil\log T\rceil)$ of $Q(u)$.

1 u samples two agents u_1 and u_2.

2 u updates $b(u)$ with the majority of $b(u)$, $b(u_1)$ and $b(u_2)$.

3 If $C' = 0$, increment Q by one unit modulo T.

Output: The clock modulo T is obtained as $C := (C' + Q\cdot T') \mod T$

Algorithm 3: The protocol 4-bit Syn-Clock, executed by each agent u.

Proof of Theorem 1.2. From Lemma 4.1, we know that Syn-Intermediate synchronizes clocks modulo T in time $\tilde{\mathcal{O}}\left(\log^2 T\log n\right)$ using only 3-bit messages, provided that T is a power of 2. While protocol Syn-Intermediate emulates protocol Syn-Simple, it displays the first bit of the message of Syn-Simple only once every $\tilde{\mathcal{O}}(\log T)$ rounds. Of course, it would be more efficient to display it $\mathcal{O}(\log n)$ times in a row, so that maj-consensus would make every agent agree on this bit, and then move to agreeing on the second bit, and so on. To achieve this, as in the proof of Syn-Simple, we can view a clock modulo T, say Q, as written on $\log T$ bits. If agents already possess a "small" counter modulo $T' := \mathcal{O}(\log T\log n)$ they can use it to display the first bit for $\mathcal{O}(\log n)$ rounds, then the second one for $\mathcal{O}(\log n)$ rounds, and so on until each one of the $\lceil\log T\rceil$ bits of T has been synchronized. This would synchronize all bits of the desired clock within $\mathcal{O}(\log T\log n)$ rounds, w.h.p., while being very economical in terms of message length, since only 1 bit is displayed at any time.

Therefore, we can use Lemma 4.1 to synchronize a counter modulo $\mathcal{O}(\log T\log n)$ in $\tilde{\mathcal{O}}((\log\log T)^2\log n)$ rounds, using 3 bits per message. Then, we can use a fourth bit to run maj-consensus on each of the $\log T$ bits of Q for $\mathcal{O}(\log n)$ consecutive rounds, for a total running time of $\mathcal{O}(\log T\log n)$ rounds. At this point, an application of the Message Reduction Theorem would give us a protocol with running time $\mathcal{O}(\log T\log n)$ using 3-bit messages. However, perhaps surprisingly, a similar strategy enables us to synchronize a clock modulo any integer (not necessarily a power of 2).

Let us assume that $T\in\mathbb{N}$ is an arbitrary integer. Let $\gamma\log n$ be an upper bound on the convergence time of maj-consensus which guarantees a correct consensus with probability at least $1-n^{-2}$, for some constant γ large enough [24]. Let T' be the smallest power of 2 bigger than $\log T\cdot(\gamma\log n+\gamma\log\log T)$. By Lemma 4.1, using 3 bits, the agents can build a synchronized clock C' running modulo T' in

time $\tilde{\mathcal{O}}((\log\log T)^2 \log n)$. The other main ingredient in this construction is another clock $Q_{T'}$ which is incremented once every T' rounds and runs modulo T. The desired clock modulo T, which we denote C, is obtained by

$$C := (C' + Q_{T'} \cdot T') \mod T.$$

It is easy to check, given the definitions of C' and $Q_{T'}$ that this choice indeed produces a clock modulo T.

It remains to show how the clock $Q_{T'}$ modulo T is synchronized. On a first glance, it may seem as if we did not simplify the problem since Q is a clock modulo T itself. However, the difference between $Q_{T'}$ and a regular clock modulo T is that $Q_{T'}$ is incremented only once every T' rounds. This is exploited as follows.

The counter $Q_{T'}$ is written on $\lceil \log T \rceil$ internal bits. We show how to synchronize $Q_{T'}$ using a 4-th bit in the messages, similarly to the aforementioned strategy to synchronize Q; we later show how to remove this assumption using the Message Reduction Theorem. Let us call a loop of C' modulo T' an *epoch*. The rounds of an epoch are divided in phases of equal length $\gamma \log n + \gamma \log\log T$ (the remaining $T' \mod (\gamma \log n + \gamma \log\log T)$ rounds are just ignored). The clock C' determines which bit from $Q_{T'}$ to display. The first bit of $Q_{T'}$ is displayed during the first phase, then the second one is displayed during the second phase, and so on. By Theorem 2.1, the length of each phase guarantees that consensus is achieved on each bit of $Q_{T'}$ via[6] MAJ-CONSENSUS w.h.p. More precisely, after the first bit has been displayed for $\gamma \log n + \gamma \log\log T$ rounds, all agents agree on it with probability[7] $1 - \frac{1}{n^2 \log T}$, provided γ is large enough. Thus, at the end of an epoch, agents agree on all $\lceil \log T \rceil$ bits of $Q_{T'}$ with probability greater than $(1 - \frac{1}{n^2 \log T})^{\log T} \gg 1 - \mathcal{O}(n^{-2})$.

We have thus shown that, by the time C' reaches its maximum value of T', i.e. after one epoch, all agents agree on $Q_{T'}$ w.h.p. and then increment it jointly. From Lemma 4.1, SYN-INTERMEDIATE takes $\tilde{\mathcal{O}}\left(\log^2 T' \log n\right) = \mathcal{O}\left((\log\log n + \log\log T)^2 \log n\right) = \mathcal{O}\left(\left((\log\log n)^2 \log n + (\log\log T)^2 \log n\right)\right)$ rounds to synchronize a clock C' modulo T' w.h.p. Together with the $\log T\,(\gamma \log n + \gamma \log\log T)$ rounds to agree on $Q_{T'}$ w.h.p., this implies that after $\log T \log n \cdot (\log\log T)^{\mathcal{O}(1)} \cdot (\log\log n)^{\mathcal{O}(1)} = \tilde{\mathcal{O}}(\log T \log n)$ rounds the clocks C are all synchronized w.h.p.

Finally, we show how to get rid of the extra 4-th bit to achieve agreement on $Q_{T'}$. Observe that, once C' is synchronized, this bit is used in a self-stabilizing way. Thus, since SYN-INTERMEDIATE has the bitwise-independence property, using Lemma 2.1, the protocol we described above possesses the bitwise-independence property too. By using the Message Reduction Theorem we can thus reduce the message size from 4 bits to $\lceil \log 4 \rceil + 1 = 3$ bits, while only incurring a constant multiplicative loss in the running time. The clock we obtain, counts modulo T but is incremented every 4 rounds only. It follows from Remark 2 that we may still view this as a clock modulo T. □

REMARK 3. (INTERNAL MEMORY SPACE) *The internal memory space needed to implement our protocols* SYN-SIMPLE, SYN-INTERMEDIATE, *and* SYN-CLOCK *is close to* $\log T$ *in all cases: protocol* SYN-SIMPLE *uses one counter written on* $\log T$ *bits,* SYN-INTERMEDIATE *needs internal memory of size*

$$\log T + \mathcal{O}(\log\log T + \log\log\log T + \ldots) \leq \log T(1 + o(1)),$$

[6] Observe that, once clock C' is synchronized, the bits of $Q_{T'}$ do not change for each agent during each subphase. Thus, we may replace MAJ-CONSENSUS by the MIN protocol where on each round of subphase i each agent u pulls another agent v u.a.r. and updates her i-th bit of Q to the minimum between her current i-th bit of Q and the one of v. However, for simplicity's sake, we reuse the already introduced MAJ-CONSENSUS protocol.

[7] From Theorem 2.1, we have that after $\gamma \log n$ rounds, with γ large enough, the probability that consensus has not been reached is smaller than $\frac{1}{n^2}$. Thus, after $N \cdot \gamma \log n$ rounds, the probability that consensus has not been reached is smaller than $\frac{1}{n^{2N}}$. If we choose $N \log n = \log n + \log\log T$, we thus get the claimed upper bound $\frac{1}{n^2 \log T}$.

and the internal memory requirement of Syn-Clock *is of order* $\log T + \log\log n$.

5 Majority Bit Dissemination with a Clock

In this section we assume that agents are equipped with a synchronized clock C modulo $\gamma \log n$ for some big enough constant $\gamma > 0$. In the previous section we showed how to establish such a synchronized clock in $\tilde{\mathcal{O}}(\log n)$ time and using 3-bit messages. We have already seen in Section 1.2 how to solve the Bit Dissemination problem (when we are promised to have a single source agent) assuming such synchronized clocks, by paying an extra bit in the message size and an $\mathcal{O}(\log n)$ additive factor in the running time. This section is dedicated to showing that, in fact, the more general Majority Bit Dissemination problem can be solved with the same time complexity and using 3-bit messages, proving Theorem 1.1.

In Section 5.1, we describe and analyze protocol Syn-Phase-Spread, which solves Majority Bit Dissemination by paying only a $\mathcal{O}(\log n)$ additive overhead in the running time w.r.t. Clock Synchronization. For clarity's sake, we first assume that the protocol is using 4 bits (i.e. 1 additional bit over the 3 bits used for Clock Synchronization), and we later show how to decrease the number of bits back to 3 in Section 5.2, by applying the Message Reduction Theorem.

The main idea behind the 3(+1)-bit protocol, called Syn-Phase-Spread, is to make the sources' input bits disseminate on the system in a way that preserves the initial ratio $\frac{k_1}{k_0}$ between the number of sources supporting the majority and minority input bit. This is achieved by dividing the dissemination process in phases, similarly to the main protocol in [33] which was designed to solve the Bit Dissemination problem in a variant of the $\mathcal{PUSH}$ model in which messages are affected by noise. The phases induce a spreading process which allows to leverage on the concentration property of the Chernoff bounds, preserving the aforementioned ratio. While, on an intuitive level, the role of noisy messages in the model considered in [33] may be related to the presence of sources having conflicting opinion in our setting, we remark that our protocol and its analysis depart from those of [33] on several key points: while the protocol in [33] needs to know the noise parameter, Syn-Phase-Spread does not assume any knowledge about the number of different sources, and our analysis does not require to control the growth of the number of speaking agents from above.

In order to perform such spreading process with 1 bit only, the protocol in [33] leverages on the fact that in the $\mathcal{PUSH}$ model agents can choose *when to speak*, i.e. whether to send a message or not. To emulate this property in the $\mathcal{PULL}$ model, we use the parity of the clock C: on odd rounds agents willing to "send" a 0 display 0, while others display 1 and conversely on even rounds. Rounds are then grouped by two, so 2 rounds in the $\mathcal{PULL}$ model correspond to 1 round in the $\mathcal{PUSH}$ version.

5.1 Protocol Syn-Phase-Spread In this section we describe protocol Syn-Phase-Spread. As mentioned above, for clarity's sake we assume that Syn-Phase-Spread uses 4-bit messages, and we show how to remove this assumption in Section 5.2. Three of such bits are devoted to the execution Syn-Clock, in order to synchronize a clock C modulo $2\lceil\gamma_{phase}\log n\rceil + \gamma_{phase}\lceil 2\log n\rceil$ for some constant γ_{phase} large enough. Throughout this section we assume, thanks to Theorem 1.2, that C has already been synchronized, which happens after $\tilde{\mathcal{O}}(\log n)$ rounds from the start of the protocol. In Section 5.1.1, we present a protocol Phase-Spread solving Majority Bit Dissemination assuming agents already share a common clock.

5.1.1 Protocol Phase-Spread Let γ_{phase} be a constant to be set later. Protocol Phase-Spread is executed periodically over periods of length $2\lceil\gamma_{phase}\log n\rceil + \gamma_{phase}\lceil 2\log n\rceil$, given by a clock C. One run of length $2\lceil\gamma_{phase}\log n\rceil + \gamma_{phase}\lceil 2\log n\rceil$ is divided in $2 + \lceil 2\log n\rceil$ phases, the first and the last ones lasting $\lceil\gamma_{phase}\log n\rceil$ rounds, all the other $\lceil 2\log n\rceil$ phases lasting γ_{phase} rounds. The first phase is called

boosting, the last one is called *polling*, and all the intermediate ones are called *spreading*. For technical convenience, in Phase-Spread agents disregard the messages they get as their second pull. In other words, Phase-Spread works in the $\mathcal{PULL}(1)$ model.

During the boosting and the spreading phases, as we already explained in the introduction of this section, we make use of the parity of time to emulate the ability to actively send a message or not to communicate anything as in the $\mathcal{PUSH}$ model(in the first case we say that the agent is *speaking*, in the second case we say that the agent is *silent*). This induces a factor 2 slowdown which we henceforth omit for simplicity.

At the beginning of the boosting, each non-source agent u is silent. During the boosting and during each spreading phase, each silent agent pulls until she sees a speaking agent. When a silent agent u sees a speaking agent v, u memorizes $b_1(v)$ but remains silent until the end of the phase; at the end of the current phase, u starts speaking and sets $b_1(u) = b_1(v)$. The bit b_1 is then never modified until the clock C reaches 0 again. Then, during the polling phase, each agent u counts how many agents with $b_1 = 1$ and how many with $b_1 = 0$ she sees. At the end of the phase, each agent u sets their output bit to the most frequent value of b_1 observed during the polling phase. We want to show that, for all agents, the latter is w.h.p. b_{maj} (the most frequent initial opinion among sources).

Phase-Spread protocol

1 If u is not speaking and $b_1(u)$ has not yet been set, and the current phase is either the boosting or the spreading one, u does the following:

2 | u observes a random agent v.

3 | If v is speaking, u sets $b_1(u)$ equal to $b_1(v)$,
and u will be speaking from the next phase.

4 | u sets c_0 and c_1 equal to 0.

5 If the current phase is polling:

6 | u observes a random agent v.

7 | If $b_1(v) = 1$, u increments c_1, otherwise increment c_0.

8 u outputs 1 if and only if $c_1 > c_0$.

Algorithm 4: The protocol Phase-Spread, executed by each agent u.

5.1.2 Analysis We prove that at the end of the last spreading phase w.h.p. all agents are speaking and each agent has $b_1 = 1$ with probability $\frac{1}{2} + \epsilon_{end}$ for some positive constant $\epsilon_{end} = \epsilon_{end}\left(\gamma_{phase}, \epsilon\right)$ (where the dependency in γ_{phase} is monotonically increasing), $b_1 = 0$ otherwise. From the Chernoff bound (Corollary A.1) and the union bound, this implies that when $\gamma_{phase} > \frac{8}{\epsilon_{end}}$ at the end of the polling phase w.h.p. each agent learns b_{maj}. Without loss of generality, let $b_{maj} = 1$, i.e. $k_1 > k_0$. For convenience, we estimate ratios of the form $\frac{k_1}{k_0}$, which requires that $k_0 > 0$. The analysis can easily be adapted to handle the case where $k_0 = 0$.

For $\epsilon \in \{0, 1\}$, let us denote $k_\epsilon^{(i)}$ the number of nodes with $b_1(\cdot) = \epsilon$ at the end of phase i. The analysis is divided in the following lemmas.

Lemma 5.1. *At the end of the boosting phase it holds w.h.p.*

$$k_1^{(1)} + k_0^{(1)} \geq \begin{cases} (k_1+k_0)\frac{\gamma_{phase}}{3}\log n & \text{if } k_1+k_0 < \frac{n}{2\gamma_{phase}\log n} \\ n\left(1-\frac{1}{\sqrt{e}}\right)+\frac{1}{\sqrt{e}}(k_1+k_0)-\sqrt{n\log n} & \text{if } \frac{n}{2\gamma_{phase}\log n} \leq k_1+k_0 \leq n-2\sqrt{n\log n}, \\ n & \text{otherwise.} \end{cases}$$

Moreover,

$$\frac{k_1^{(1)}}{k_0^{(1)}} \geq \frac{k_1}{k_0}\cdot\left(1-\sqrt{\frac{9}{\gamma_{phase}k_0}}\right). \tag{5.2}$$

Proof. By using Fact 4.1, we have

$$\begin{aligned} \mathbb{E}\left[k_1^{(1)}+k_0^{(1)}\right] &= k_1+k_0+(n-k_1-k_0)\left(1-\left(1-\frac{k_1+k_0}{n}\right)^{\gamma_{phase}\log n}\right) \\ &\geq k_1+k_0+(n-k_1-k_0)\left(1-e^{-\frac{k_1+k_0}{n}\gamma_{phase}\log n}\right). \end{aligned} \tag{5.3}$$

We distinguish three cases.

Case $k_1+k_0 < \frac{n}{2\gamma_{phase}\log n}$**.** By using Fact 4.1 again, from (5.3) we get

$$\begin{aligned} \mathbb{E}\left[k_1^{(1)}+k_0^{(1)}\right] &\geq k_1+k_0+(n-k_1-k_0)\left(1-e^{-\frac{k_1+k_0}{n}\gamma_{phase}\log n}\right) \\ &\geq k_1+k_0+(n-k_1-k_0)\frac{\frac{k_1+k_0}{n}\gamma_{phase}\log n}{1+\frac{k_1+k_0}{n}\gamma_{phase}\log n} \\ &\geq k_1+k_0+(n-k_1-k_0)\frac{k_1+k_0}{n}\frac{\gamma_{phase}}{2}\log n \\ &\geq k_1+k_0+\left(1-\frac{k_1+k_0}{2n}\right)(k_1+k_0)\frac{\gamma_{phase}}{2}\log n \\ &\geq (k_1+k_0)\left(1+\left(1-\frac{1}{4\gamma_{phase}\log n}\right)\frac{\gamma_{phase}}{2}\log n\right) \\ &\geq (k_1+k_0)\frac{\gamma_{phase}}{2}\log n. \end{aligned} \tag{5.4}$$

From the Chernoff bound (Lemma A.1), we thus get that w.h.p.

$$k_1^{(1)}+k_0^{(1)} \geq (k_1+k_0)\frac{\gamma_{phase}}{3}\log n.$$

Case $\frac{n}{2\gamma_{phase}\log n} \leq k_1+k_0 \leq n-2\sqrt{n\log n}$**.** From (5.3), we have

$$\begin{aligned} \mathbb{E}\left[k_1^{(1)}+k_0^{(1)}\right] &\geq k_1+k_0+(n-k_1-k_0)\left(1-e^{-\frac{k_1+k_0}{n}\gamma_{phase}}\right) \\ &\geq k_1+k_0+(n-k_1-k_0)\left(1-\frac{1}{\sqrt{e}}\right) \\ &\geq n\left(1-\frac{1}{\sqrt{e}}\right)+\frac{k_1+k_0}{\sqrt{e}}. \end{aligned}$$

From the Chernoff bound (Lemma A.1), we thus get that w.h.p.

$$k_1^{(1)} + k_0^{(1)} \geq n\left(1 - \frac{1}{\sqrt{e}}\right) + \frac{k_1 + k_0}{\sqrt{e}} - \sqrt{n \log n}.$$

Case $k_1^{(1)} + k_0^{(1)} > n - 2\sqrt{n \log n}$. The probability that a silent agent does not observe a speaking one is

$$\left(\frac{n - k_1 - k_0}{n}\right)^{\gamma_{phase} \log n} \leq \left(\frac{4 \log n}{n}\right)^{\frac{1}{2}\gamma_{phase} \log n},$$

hence by a simple union bound it follows that w.h.p. all agents are speaking.

Now, we prove (5.2). As before, we have two cases. The first case, $\frac{k_1}{k_0} \geq \frac{n}{2\gamma_{phase} \log n}$, is a simple consequence of the Chernoff bound (Lemma A.1).

In the second case, $\frac{k_1}{k_0} < \frac{n}{2\gamma_{phase} \log n}$, let us consider the set of agents S_{boost} that start speaking at the end of the boosting, i.e. that observe a speaking agent during the phase. Observe that $|S_{boost}| = k_1^{(1)} - k_1 + k_0^{(1)} - k_0$. The probability that an agent in S_{boost} observes a source supporting 1 (resp. 0) is $\frac{k_1}{k_1 + k_0}$ (resp. $\frac{k_0}{k_1 + k_0}$). Thus

$$\begin{aligned} \mathbb{E}\left[k_1^{(1)}\right] &= k_1 + \frac{k_1}{k_1 + k_0}\mathbb{E}\left[|S_{boost}|\right] \quad \text{and} \\ \mathbb{E}\left[k_0^{(1)}\right] &= k_0 + \frac{k_0}{k_1 + k_0}\mathbb{E}\left[|S_{boost}|\right]. \end{aligned} \tag{5.5}$$

In particular

$$\frac{\mathbb{E}\left[k_1^{(1)}\right]}{\mathbb{E}\left[k_0^{(1)}\right]} = \frac{k_1 + \frac{k_1}{k_1 + k_0}\mathbb{E}\left[|S_{boost}|\right]}{k_0 + \frac{k_0}{k_1 + k_0}\mathbb{E}\left[|S_{boost}|\right]} = \frac{k_1}{k_0}, \tag{5.6}$$

and from (5.4) and (5.5) we have

$$\begin{aligned} \mathbb{E}\left[k_0^{(1)}\right] &\geq \frac{k_0}{k_1 + k_0}\mathbb{E}\left[|S_{boost}|\right] \\ &= \frac{k_0}{k_1 + k_0}\left(\mathbb{E}\left[k_1^{(1)} + k_0^{(1)}\right] - (k_1 + k_0)\right) \\ &\geq (1 - o(1))\frac{k_0}{k_1 + k_0}\frac{\gamma_{phase}}{2}(k_1 + k_0)\log n \\ &= (1 - o(1))k_0\frac{\gamma_{phase}}{2}\log n, \end{aligned} \tag{5.7}$$

where the lower bound follows from the assumption $\frac{k_1}{k_0} < \frac{n}{2\gamma_{phase} \log n}$ and (5.4). From (5.7) and the multiplicative form of the Chernoff bound (Corollary A.1), we have that w.h.p.

$$\begin{aligned} k_1^{(1)} &\geq \mathbb{E}\left[k_1^{(1)}\right] - \sqrt{\mathbb{E}\left[k_1^{(1)}\right]\log n} \quad \text{and} \\ k_0^{(1)} &\leq \mathbb{E}\left[k_0^{(1)}\right] + \sqrt{\mathbb{E}\left[k_0^{(1)}\right]\log n}. \end{aligned} \tag{5.8}$$

Thus, since (5.5) implies $\mathbb{E}\left[k_1^{(1)}\right] \geq \mathbb{E}\left[k_0^{(1)}\right]$, we have

$$\begin{aligned}
\frac{k_1^{(1)}}{k_0^{(1)}} &\geq \frac{\mathbb{E}\left[k_1^{(1)}\right] - \sqrt{\mathbb{E}\left[k_1^{(1)}\right]\log n}}{\mathbb{E}\left[k_0^{(1)}\right] + \sqrt{\mathbb{E}\left[k_0^{(1)}\right]\log n}} \\
&= \frac{\mathbb{E}\left[k_1^{(1)}\right]}{\mathbb{E}\left[k_0^{(1)}\right]} \cdot \frac{1 - \sqrt{\frac{\log n}{\mathbb{E}\left[k_1^{(1)}\right]}}}{1 + \sqrt{\frac{\log n}{\mathbb{E}\left[k_0^{(1)}\right]}}} \\
&\geq \frac{\mathbb{E}\left[k_1^{(1)}\right]}{\mathbb{E}\left[k_0^{(1)}\right]} \cdot \left(1 - \sqrt{\frac{\log n}{\mathbb{E}\left[k_1^{(1)}\right]}} - \sqrt{\frac{\log n}{\mathbb{E}\left[k_0^{(1)}\right]}}\right) \\
&\geq \frac{\mathbb{E}\left[k_1^{(1)}\right]}{\mathbb{E}\left[k_0^{(1)}\right]} \cdot \left(1 - 2\sqrt{\frac{\log n}{\mathbb{E}\left[k_0^{(1)}\right]}}\right) \\
&= \frac{k_1}{k_0} \cdot \left(1 - \sqrt{\frac{9}{k_0 \gamma_{phase}}}\right), \qquad (5.9)
\end{aligned}$$

concluding the proof.

Lemma 5.2. *At the end of the $i+1$th spreading phase, the following holds w.h.p.*

$$k_1^{(i+1)} + k_0^{(i+1)} \geq \begin{cases} \left(k_1^{(i)} + k_0^{(i)}\right)\frac{\gamma_{phase}}{3}, \text{ if } k_1^{(i)} + k_0^{(i)} < \frac{n}{2\gamma_{phase}} \\ n\left(1 - \frac{1}{\sqrt{e}}\right) + \frac{1}{\sqrt{e}}\left(k_1^{(i)} + k_0^{(i)}\right) - \sqrt{n\log n}, \text{ if } \frac{n}{2\gamma_{phase}} \leq k_1^{(i)} + k_0^{(i)} \leq n - 2\sqrt{n\log n} \\ n, \text{ otherwise.} \end{cases}$$

$$\frac{k_1^{(i+1)}}{k_0^{(i+1)}} \geq \frac{k_1^{(i)}}{k_0^{(i)}}\left(1 - 4\sqrt{\frac{\log n}{\gamma_{phase} k_0^{(i)}}}\right). \qquad (5.10)$$

Proof. The proof is almost the same as that of Lemma 5.1. Thus, we condense some analogous calculations.

By using Fact 4.1, we have

$$\mathbb{E}\left[k_1^{(i+1)} + k_0^{(i+1)}\right] \geq k_1^{(i)} + k_0^{(i)} + \left(n - k_1^{(i)} - k_0^{(i)}\right)\left(1 - e^{-\frac{k_1^{(i)} + k_0^{(i)}}{n}\gamma_{phase}}\right). \qquad (5.11)$$

We distinguish three cases.

Case $k_1^{(i)} + k_0^{(i)} < \frac{n}{2\gamma_{phase}}$. By using Fact 4.1 again, from (5.11) we get

$$\mathbb{E}\left[k_1^{(i+1)} + k_0^{(i+1)}\right] \geq k_1^{(i)} + k_0^{(i)} + \left(n - k_1^{(i)} - k_0^{(i)}\right) \cdot \frac{k_1^{(i)} + k_0^{(i)}}{2n}\gamma_{phase} \geq \left(k_1^{(i)} + k_0^{(i)}\right)\frac{\gamma_{phase}}{2}. \qquad (5.12)$$

After the boosting phase, i.e. for $i \geq 1$, it follows from Lemma 5.1 that $k_1^{(i)} + k_0^{(i)} = \Omega\left(\gamma_{phase} \log n\right)$. From the Chernoff bound (Lemma A.1), if γ_{phase} is chosen big enough, we thus get that w.h.p.

$$k_1^{(i+1)} + k_0^{(i+1)} \geq \left(k_1^{(i)} + k_0^{(i)}\right) \frac{\gamma_{phase}}{3}.$$

Case $\frac{n}{2\gamma_{phase}} \leq k_1^{(i)} + k_0^{(i)} \leq n - 2\sqrt{n \log n}$**.** From (5.11), we have

$$\mathbb{E}\left[\left(k_1^{(i+1)} + k_0^{(i+1)}\right)\right] \geq k_1^{(i)} + k_0^{(i)} + \left(n - k_1^{(i)} - k_0^{(i)}\right)\left(1 - \frac{1}{\sqrt{e}}\right) \geq n\left(1 - \frac{1}{\sqrt{e}}\right) + \frac{1}{\sqrt{e}}\left(k_1^{(i)} + k_0^{(i)}\right).$$

From the Chernoff bound (Lemma A.1), we thus get that w.h.p.

$$k_1^{(i+1)} + k_0^{(i+1)} \geq n\left(1 - \frac{1}{\sqrt{e}}\right) + \frac{1}{\sqrt{e}}\left(k_1^{(i)} + k_0^{(i)}\right) - \sqrt{n \log n}.$$

Case $k_1^{(i)} + k_0^{(i)} > n - 2\sqrt{n \log n}$**.** The probability that a silent agent does not observe a speaking one is

$$\left(\frac{n - k_1^{(i)} - k_0^{(i)}}{n}\right)^{\gamma_{phase}} \leq \left(\frac{4 \log n}{n}\right)^{\frac{1}{2}\gamma_{phase}},$$

hence by a simple union bound it follows that w.h.p. all agents are speaking.

Now, we prove (5.10). As in the proof of (5.2), we have two cases. The first case, $\frac{k_1}{k_0} \geq \frac{n}{2\gamma_{phase}}$, is a simple consequence of the Chernoff bound (Lemma A.1). Otherwise, let us assume $\frac{k_1}{k_0} < \frac{n}{2\gamma_{phase}}$. With an analogous argument to that for (5.5) and (5.6) we can prove

$$\frac{\mathbb{E}\left[k_1^{(i+1)}\right]}{\mathbb{E}\left[k_0^{(i+1)}\right]} = \frac{k_1^{(i)}}{k_0^{(i)}}, \tag{5.13}$$

and

$$\begin{aligned}
\mathbb{E}\left[k_1^{(i+1)}\right] &= k_1^{(i)} + \frac{k_1^{(i)}}{k_1^{(i)} + k_1^{(i)}} \mathbb{E}\left[k_1^{(i+1)} - k_1^{(i)} + k_0^{(i+1)} - k_0^{(i)}\right], \\
\mathbb{E}\left[k_0^{(i+1)}\right] &= k_0^{(i)} + \frac{k_0^{(i)}}{k_1^{(i)} + k_0^{(i)}} \mathbb{E}\left[k_1^{(i+1)} - k_1^{(i)} + k_0^{(i+1)} - k_0^{(i)}\right]. \qquad (5.14)
\end{aligned}$$

As in (5.8), from the multiplicative form of the Chernoff bound (Corollary A.1) we have that w.h.p.

$$\begin{aligned}
k_1^{(i+1)} &\geq \mathbb{E}\left[k_1^{(i+1)}\right] - \sqrt{\mathbb{E}\left[k_1^{(i+1)}\right] \log n} \quad \text{and} \\
k_0^{(i+1)} &\leq \mathbb{E}\left[k_0^{(i+1)}\right] + \sqrt{\mathbb{E}\left[k_0^{(i+1)}\right] \log n}. \qquad (5.15)
\end{aligned}$$

Thus, as in (5.9), from (5.15) and (5.13), we get

$$\frac{k_1^{(i+1)}}{k_0^{(i+1)}} \geq \frac{\mathbb{E}\left[k_1^{(i+1)}\right]}{\mathbb{E}\left[k_0^{(i+1)}\right]} \cdot \left(1 - 2\sqrt{\frac{\log n}{\mathbb{E}\left[k_0^{(i+1)}\right]}}\right) \geq \frac{k_1^{(i)}}{k_0^{(i)}} \cdot \left(1 - 4\sqrt{\frac{\log n}{\gamma_{phase} k_0^{(i)}}}\right),$$

where, as in (5.7), in the last inequality we used that from (5.12) and (5.14) it holds $\mathbb{E}\left[k_0^{(i+1)}\right] \geq \frac{\gamma_{phase}}{4} k_0^{(i)}$.

From the previous two lemmas, we can derive the following corollary, which concludes the proof.

COROLLARY 5.1. *If $k_1 \geq k_0(1+\epsilon)$ for some constant $\epsilon > 0$, then at the end of the last spreading phase it holds w.h.p.*

$$k_1^{(1+2\log n)} = n - k_0^{(1+2\log n)} \geq k_0^{(1+2\log n)}(1+\epsilon_{end}), \tag{5.16}$$

where $\epsilon_{end} = \frac{\epsilon}{2} - \frac{4}{\sqrt{\gamma_{phase}}}$.

Proof. The equality in (5.16) follows from the first part of Lemma 5.1. When $k_1^{(i)} + k_0^{(i)} < \frac{n}{2\gamma_{phase}}$, $k_1^{(i)} + k_0^{(i)}$ increases by multiplicative a factor γ_{phase} at the end of each spreading phase. When $\frac{n}{2\gamma_{phase}} \leq k_1^{(i)} + k_0^{(i)} \leq n - 2\sqrt{n\log n}$,

$$n - k_1^{(i+1)} - k_0^{(i+1)} \leq \frac{n - k_1^{(i)} - k_0^{(i)}}{\sqrt{e}} - \sqrt{n\log n} \leq \frac{n - k_1^{(i)} - k_0^{(i)}}{\sqrt{e}}.$$

Hence the number of silent agents decreases by a factor $\sqrt{e}$ after each spreading phase. Lastly, when $k_1^{(i)} + k_0^{(i)} > n - 2\sqrt{n\log n}$, after one more spreading phase, a simple application of the union bound shows that $k_1^{(i+1)} + k_0^{(i+1)}$ is equal to n w.h.p. As a consequence, if γ_{phase} is big enough, after less than $1 + 2\log n$ spreading phases w.h.p it holds that $k_1^{(1+2\log n)} = n - k_0^{(1+2\log n)}$.

The inequality in (5.16) can be derived from (5.10), as follows. From (5.2) and (5.10) we have

$$\frac{k_1^{(1+2\log n)}}{k_0^{(1+2\log n)}} \geq \frac{k_1}{k_0}\left(1 - \sqrt{\frac{9}{\gamma_{phase}k_0}}\right)\prod_{i=2}^{1+2\log n}\left(1 - \sqrt{\frac{16\log n}{\gamma_{phase}k_0^{(i)}}}\right). \tag{5.17}$$

We can estimate the product as

$$\begin{aligned}
\prod_{i=2}^{1+2\log n}\left(1 - \sqrt{\frac{16\log n}{\gamma_{phase}k_0^{(i)}}}\right) &\geq \exp\left(-4\sum_{i=2}^{1+2\log n}\frac{1}{\left(\sqrt{\gamma_{phase}}\right)^i}\right) \\
&\geq \exp\left\{4\left(1 + \frac{1}{\sqrt{\gamma_{phase}}} - \frac{1 - \left(\gamma_{phase}\right)^{-\frac{2+2\log n}{2}}}{1 - \left(\gamma_{phase}\right)^{-\frac{1}{2}}}\right)\right\} \\
&\geq \exp\left\{-4\left(\frac{1}{\gamma_{phase} - \sqrt{\gamma_{phase}}} - n^{-\frac{2\log\gamma_{phase}}{2}}\right)\right\} \\
&\geq \exp\left\{-\frac{4}{\gamma_{phase}}\right\} \\
&\geq \left(1 - \frac{5}{\gamma_{phase}}\right). \qquad (5.18)
\end{aligned}$$

In the first inequality we used that $1 - x \geq e^{-x}$ if $|x| < 1$. To go from the first to the second line, we use $\sum_2^a x = -1 - x + \frac{1-x^{a+1}}{1-x}$, with $a = 1 + 2\log n$ and $x = \frac{1}{\left(\sqrt{\gamma_{phase}}\right)^i}$. To go from the third to the fourth line, we use that for n big enough, $\frac{1}{\gamma_{phase} - \sqrt{\gamma_{phase}}} - n^{-\frac{2\log\gamma_{phase}}{2}} \geq \frac{1}{\gamma_{phase}}$. The last inequality is obtained from $e^{-x} \geq 1 - \frac{5}{4}x$, if $x > 0$ is small enough.

Finally, from (5.17) and (5.18) we get

$$\frac{k_1^{(1+2\log n)}}{k_0^{(1+2\log n)}} \geq \frac{k_1}{k_0}\left(1-\sqrt{\frac{9}{\gamma_{phase}k_0}}\right)\left(1-\frac{5}{\gamma_{phase}}\right) \geq \frac{k_1}{k_0}\left(1-\frac{4}{\sqrt{\gamma_{phase}}}\right),$$

which, together with the hypothesis $\frac{k_1}{k_0} \geq 1+\epsilon$, concludes the proof.

5.2 Proof of Theorem 1.1

Proof. From Corollary 5.1, it follows that at the end of the last spreading phase, all agents have been informed. After the last spreading phase, during the polling phase, each agent samples $\gamma_{phase}\log n$ opinions from the population and then adopts the majority of these as her output bit. Thus, (5.16) ensures that each sample holds the correct opinion with probability $\geq \frac{1}{2}+\epsilon_{end}$. Hence, by the Chernoff bound and a union bound, if γ_{phase} is big enough then the majority of the $\gamma_{phase}\log n$ samples corresponds to the correct value for all the n agents w.h.p.

The protocol obtained so far solves Majority Bit Dissemination, but it does it using 4 bits per message rather than 3. Indeed, synchronizing a clock using Syn-Clock takes 3 bits, and we use an extra bit to execute Phase-Spread described in Section 5.1.1. However, the protocol Syn-Phase-Spread has the bitwise-independence property. This follows from Lemma 2.1 with Syn-Generic =Syn-Clock, P =Phase-Spread, Syn-P =Syn-Phase-Spread, together with the observation that Phase-Spread is self-stabilizing. We can thus reduce the message length of Syn-Phase-Spread to 3 bits using again the Message Reduction Theorem, with a time overhead of a factor 4 only.

6 Conclusion and Open Problems

This paper deals with the construction of protocols in highly congested stochastic interaction patterns. Corresponding challenges are particularly evident when it is difficult to guarantee synchronization, which seems to be essential for emulating a typical protocol that relies on many bits per message with a protocol that uses fewer bits. Our paper shows that in the $\mathcal{PULL}$ model, if a self-stabilizing protocol satisfies the bitwise-independence property then it can be emulated with only 3 bits per message. Using this rather general transformer, we solve the self-stabilizing Clock-Synchronization and Majority Bit Dissemination problems in almost-logarithmic time and using only 3 bits per message. It remains an open problem whether the message size of either one of these problems can be further reduced while keeping the running time polylogarithmic.

In particular, even for the self-stabilizing Bit Dissemination problem (with a single source) it remains open whether there exists a polylogarithmic protocol that uses a single bit per interaction. In fact, we investigated several candidate protocols which seem promising in experimental simulation, but appear to be out of reach of current techniques for analysing randomly-interacting agent systems in a self-stabilizing context. Let us informally present one of them, called BFS[8]. Let ℓ,$k \in \mathbb{N}$ be two parameters, say of order $O(\log n)$. Agents can be in 3 states: *boosting*, *frozen* or *sensitive*. Boosting agents behave as in the maj-consensus protocol: they apply the majority rule to the 2 values they see in a given round and make it into their opinion for the next round. They also keep a counter T. If they have seen only agents of a given color b for ℓ rounds, they become sensitive to the opposite value. b-sensitive agents turn into frozen-b agents if they see value b. b-frozen agents keep the value b for k rounds before becoming boosters again. Intuitively what we expect is that, from every configuration, at some point almost all agents would be in the boosting state. Then, the boosting behavior would lead the agents to converge to a value b (which depends on the initial conditions). Most agents would then

[8] A similar protocol was suggested during discussions with Bernhard Haeupler.

become sensitive to $1-b$. If the source has opinion $1-b$ then there should be a "switch" from b to $1-b$. The "frozen" period is meant to allow for some delay in the times at which agents become sensitive, and then flip their opinion.

Acknowledgments: The problem of self-stabilizing Bit Dissemination was introduced through discussions with Ofer Feinerman. The authors are also thankful for Omer Angel, Bernhard Haeupler, Parag Chordia, Iordanis Kerenidis, Fabian Kuhn, Uri Feige, and Uri Zwick for helpful discussions regarding that problem. The authors also thank Michele Borassi for his helpful suggestions regarding the Clock Synchronization problem.

References

[1] Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-joseph. A biological solution to a fundamental distributed computing problem. *Science*, 2011.

[2] D. Alistarh and R. Gelashvili. Polylogarithmic-time leader election in population protocols. In *ICALP*, pages 479–491, 2015.

[3] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[4] D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

[5] D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang. Self-stabilizing population protocols. *TAAS*, 3(4), 2008.

[6] D. Angluin, M. J. Fischer, and H. Jiang. *Stabilizing Consensus in Mobile Networks*, pages 37–50. Springer, 2006.

[7] J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the EATCS*, 93:98–117, 2007.

[8] H. Attiya, A. Herzberg, and S. Rajsbaum. Optimal clock synchronization under different delay assumptions. *SIAM J. Comput.*, 25(2):369–389, 1996.

[9] J. Beauquier, J. Burman, and S. Kutten. A self-stabilizing transformer for population protocols with covering. *Theor. Comput. Sci.*, 412(33):4247–4259, 2011.

[10] L. Becchetti, A. Clementi, E. Natale, F. Pasquale, and G. Posta. Self-stabilizing repeated balls-into-bins. In *SPAA*, pages 332–339, 2015.

[11] L. Becchetti, A. E. F. Clementi, E. Natale, F. Pasquale, and R. Silvestri. Plurality consensus in the gossip model. In *SODA*, pages 371–390, 2015.

[12] L. Becchetti, A. E. F. Clementi, E. Natale, F. Pasquale, and L. Trevisan. Stabilizing consensus with many opinions. In *SODA*, pages 620–635, 2016.

[13] M. Ben-Or, D. Dolev, and E. N. Hoch. Fast self-stabilizing byzantine tolerant digital clock synchronization. In *PODC*, pages 385–394, 2008.

[14] L. Boczkowski, A. Korman, and E. Natale. Brief announcement: Self-stabilizing clock synchronization with 3-bit messages. In *PODC*, 2016.

[15] L. Boczkowski, A. Korman, and E. Natale. Minimizing message size in stochastic communication patterns: Fast self-stabilizing protocols with 3 bits. In *SODA*, 2017.

[16] K. Censor-Hillel, B. Haeupler, J. A. Kelner, and P. Maymounkov. Global computation in a poorly connected world: fast rumor spreading with no dependence on conductance. In *STOC*, pages 961–970, 2012.

[17] H.-L. Chen, R. Cummings, D. Doty, and D. Soloveichik. Speed faults in computation by chemical reaction networks. *In Distributed Computing*, pages 16–30, 2014.

[18] F. Chierichetti, S. Lattanzi, and A. Panconesi. Rumor spreading in social networks. In *ICALP*, pages 375–386, 2009.

[19] C. Cooper, R. Elsässer, T. Radzik, N. Rivera, and T. Shiraga. Fast consensus for voting on general expander graphs. In *DISC*, pages 248–262. Springer, 2015.

[20] I. Couzin, J. Krause, N. Franks, and S. Levin. Effective leadership and decision making in animal groups on the move. *Nature 433*, pages 513–516, 2005.

[21] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. E. Sturgis, D. C. Swinehart, and D. B. Terry. Epidemic algorithms for replicated database maintenance. *Operating Systems Review*, 22(1):8–32, 1988.
[22] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
[23] B. Doerr and M. Fouz. Asymptotically optimal randomized rumor spreading. *Electronic Notes in Discrete Mathematics*, 38:297–302, 2011.
[24] B. Doerr, L. A. Goldberg, L. Minder, T. Sauerwald, and C. Scheideler. Stabilizing consensus with the power of two choices. In *SPAA*, pages 149–158, 2011.
[25] D. Dolev and E. N. Hoch. On self-stabilizing synchronous actions despite byzantine attacks. In *DISC*, pages 193–207, 2007.
[26] D. Dolev, J. H. Korhonen, C. Lenzen, J. Rybicki, and J. Suomela. Synchronous counting and computational algorithm design. In *SSS*, pages 237–250, 2013.
[27] S. Dolev. Possible and impossible self-stabilizing digital clock synchronization in general graphs. *Real-Time Systems*, 12(1):95–107, 1997.
[28] S. Dolev and J. L. Welch. Self-stabilizing clock synchronization in the presence of byzantine faults. *J. ACM*, 51(5):780–799, 2004.
[29] D. Doty and D. Soloveichik. Stable leader election in population protocols requires linear time. *CoRR*, abs/1502.04246, 2015.
[30] R. Elsässer, T. Friedetzky, D. Kaaser, F. Mallmann-Trenn, and H. Trinker. Efficient k-party voting with two choices. *CoRR*, abs/1602.04667, 2016.
[31] R. Elsässer and T. Sauerwald. On the runtime and robustness of randomized broadcasting. *Theor. Comput. Sci.*, 410(36):3414–3427, 2009.
[32] Y. Emek and R. Wattenhofer. Stone age distributed computing. In *PODC*, pages 137–146, 2013.
[33] O. Feinerman, B. Haeupler, and A. Korman. Breathe before speaking: efficient information dissemination despite noisy, limited and anonymous communication. In *PODC*, pages 114–123, 2014.
[34] O. Feinerman and A. Korman. Clock synchronization and estimation in highly dynamic networks: An information theoretic approach. In *SIROCCO*, pages 16–30, 2015.
[35] O. Feinerman and A. Korman. Individual versus collective cognition in social insects. *Submitted to Journal of Experimental Biology*, 2016.
[36] R. Harkness and N. Maroudas. Central place foraging by an ant (*cataglyphis bicolor* fab.): a model of searching. *Animal Behavior 33(3)*, pages 916–928, 1985.
[37] T. Herman. Phase clocks for transient fault repair. *IEEE Trans. Parallel Distrib. Syst.*, 11(10):1048–1057, 2000.
[38] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *FOCS*, pages 565–574, 2000.
[39] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS*, pages 482–491. IEEE, 2003.
[40] A. Kravchik and S. Kutten. Time optimal synchronous self stabilizing spanning tree. In Y. Afek, editor, *DISC*, *Jerusalem*, *Israel*, *October 14-18*, *2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2013.
[41] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
[42] C. Lenzen, T. Locher, P. Sommer, and R. Wattenhofer. Clock synchronization: Open problems in theory and practice. In *SOFSEM*, pages 61–70, 2010.
[43] C. Lenzen, T. Locher, and R. Wattenhofer. Tight bounds for clock synchronization. *J. ACM*, 57(2), 2010.
[44] C. Lenzen and J. Rybicki. Efficient counting with optimal resilience. In *DISC*, pages 16–30, 2015.
[45] C. Lenzen, J. Rybicki, and J. Suomela. Towards optimal synchronous counting. In *PODC*, pages 441–450, 2015.
[46] C. McDiarmid. *Concentration*, pages 195–248. Springer, 1998.
[47] N. Razin, J. Eckmann, and O. Feinerman. Desert ants achieve reliable recruitment across noisy interactions. *Journal of the Royal Society Interface; 10(20170079).*, 2013.
[48] G. Roberts. Why individual vigilance increases as group size increases. *Animal Behaviour 51*, pages 1077–

1086, 1996.
[49] D. J. Sumpter et al. Consensus decision making by fish. *Current Biology 22(25)*, pages 1773–1777, 2008.

A Technical Tools

THEOREM A.1. ([46]) *Let $X_1, ..., X_n$ be n independent random variables. If $X_i \leq M$ for each i, then*

$$\Pr\left(\sum_i X_i \geq \mathbb{E}\left[\sum_i X_i\right] + \lambda\right) \leq e^{-\frac{\lambda^2}{2\left(\sqrt{\sum_i \mathbb{E}\left[X_i^2\right]} + \frac{M\lambda}{3}\right)}}. \tag{A.1}$$

COROLLARY A.1. *Let $\mu = \mathbb{E}[\sum_i X_i]$. If the X_is are binary then, for $\lambda = \sqrt{\mu \log n}$ and sufficiently large n,* (A.1) *gives*

$$\Pr\left(\sum_i X_i \geq \mu + \sqrt{\mu \log n}\right) \leq e^{-\sqrt{\mu \log n}},$$
$$\Pr\left(\sum_i X_i \leq \mu - \sqrt{\mu \log n}\right) \leq e^{-\sqrt{\mu \log n}}.$$

Proof. The fact that the X_is are binary implies that $\sum_i \mathbb{E}\left[X_i^2\right] \leq \sum_i \mathbb{E}[X_i]$. By setting $\lambda = \sqrt{\mathbb{E}[\sum_i X_i] \log n}$, one can show that the l.h.s. of (A.1) is upper bounded by $e^{-\sqrt{\mu \log n}}$.

B Proof of Lemma B.1

LEMMA B.1. *Let $f, \tau : \mathbb{R}_+ \to \mathbb{R}$ be functions defined by $f(x) = \lceil \log x \rceil + 1$ and*

$$\tau(x) = \inf\left\{k \in \mathbb{N} \mid f^{\circledast k}(x) \leq 3\right\},$$

where we denote by $f^{\circledast k}$ the k-fold iteration of f. It holds that

$$\tau(T) \leq \log^{\circledast 4} T + \mathcal{O}(1).$$

Proof. We can notice that $f(T) \leq T - 1$, if T is bigger than some constant c. Moreover, when $f(x) \leq c$, the number of iterations before reaching 1 is $O(1)$. This implies that $\tau(T) \leq T + \mathcal{O}(1)$. But in fact, by definition, $\ell(T) = \tau\left(f^{\circledast 4}(T)\right) + 4$ (provided $f^{\circledast 4}(T) > 1$, which holds if T is big enough). Hence

$$\tau(T) \leq g\left(f^{\circledast 4}(T)\right) + 4 \leq f^{\circledast 4}(T) + \mathcal{O}(1) \leq \log^{\circledast 4} T + \mathcal{O}(1).$$